M. Carmen Ruiz Delgado

LIMITACIÓN DE RECURSOS EN PARALELISMO REAL

I.S.B.N. Ediciones de la UCLM
978-84-8427-676-0

Ediciones de la Universidad
de Castilla-La Mancha

Cuenca, 2009

UNIVERSIDAD DE CASTILLA-LA MANCHA

Departamento de Sistemas Informáticos

Limitación de Recursos en Paralelismo Real

Tesis Doctoral

Presentada por:
M. Carmen Ruiz Delgado
Dirigida por
Fernando Cuartero Gómez
Diego Cazorla López

Limitación de Recursos en Paralelismo Real

M. Carmen Ruiz Delgado

Tesis Doctoral presentada al Departamento de
Sistemas Informáticos de la Universidad de Castilla-La Mancha
para la obtención del grado de Doctor en Informática

A Carlos y María

Ella siempre decía que yo haría cosas importantes; yo siempre me reía por esa fe ciega que sólo da el amor, y le decía que sin ninguna duda se lo dedicaría. Como siempre, haciendo uso de esa sabiduría que sólo dan los años, esta vez también acertó: dos veces en mi vida he hecho algo realmente importante, pero, desgraciadamente, a los niños no se les puede poner una dedicatoria.

No creo que con su extrema humildad me hubiera permitido dedicarle ni estas dos líneas. Pero ya no está. Por eso, aunque esta tesis está dedicada a mis hijos, me gustaría recordar a alguien que dominaba el arte de la conversación y la particular disciplina de la risa, hizo mi vida muy feliz y me enseñó que cada nuevo día es algo maravilloso.

A María Pérez

Quisiera dejar por escrito...

... mi más sincero agradecimiento a mis AMIGOS (con mayúsculas) por que todos y cada uno de ellos colaboran en que mi vida sea mucho mejor. Más concretamente...

> ... a los amigos que han ejercido a las mil maravillas de directores en esta tesis, ayudándome y haciéndome sentir parte de un equipo.
>
> ... a los amigos que me han ayudado y animado en distintas etapas de mi tesis.
>
> ... a los amigos que han hecho un gran esfuerzo al intentar entender qué es *eso* sobre lo que escribo.
>
> ... a los amigos que han robado todo el tiempo que han podido de mi trabajo para hacer cosas *más importantes*.
>
> ... a los amigos, hoy compañeros, que un día fueron mis profesores y me iniciaron en esto de la informática y, muy en particular, al director de mi primer proyecto fin de carrera, que se atrevió a ser el director de mi segundo proyecto fin de carrera y que definitivamente me hizo caer en estas cosas que ahora escribo.
>
> ... a los amigos con los que comparto risas.
>
> ... a los amigos con los que comparto lágrimas.

... gracias a los miembros del tribunal que, aún estando realmente ocupados, han encontrado tiempo para participar en la lectura de esta tesis.

... gracias a mis padres por darme algo tan maravilloso como difícil de conceder, especialmente a las personas que queremos: la libertad.

... gracias a mi M. Pilar por recordarme con su sola presencia que casi todo se puede lograr.

... gracias a Josefa Delgado que con su forma de vivir me hace pensar que todavía existe la honradez, el altruismo y la bondad.

... gracias a un maravilloso ser humano que desde hace muchos años me acompaña, estimula, comprende y, sobre todo, me quiere con un amor de verdad. El que siempre, siempre está. Gracias Jesús.

... gracias a Carlos y María por existir.

Índice general

1. Introducción **1**

2. Lenguajes Formales de Especificación de Sistemas Concurrentes **9**

2.1. Objetivos de los lenguajes formales de especificación 10

2.2. Ingredientes de los lenguajes 12

2.3. Álgebra de procesos CSP 16

2.4. Extensiones de los lenguajes formales de especificación 23

2.4.1. Extensiones probabilistas 24

2.4.2. Extensiones temporales 26

2.4.3. Extensiones estocásticas 28

3. Álgebra de Procesos Temporizada BTC **33**

3.1. Justificación 33

3.2. Sintaxis del álgebra de procesos **BTC** 38

3.3. Semántica Operacional 41

3.4. Evaluación de prestaciones 46

3.5. Un ejemplo: Suma de matrices 48

3.6. **BTC** con Recursos Heterogéneos 52

3.7. Qué hemos mejorado 55

3.8. **BTC** con Recursos Expropiativos y no Expropiativos 58

3.8.1. Semántica operacional 63
3.8.2. Ejemplo: Comedor 67

4. Aplicación: Protocolo SET **71**
4.1. Qué es SET 74
4.1.1. Por qué SET 75
4.1.2. Cómo SET protege sus transacciones 75
4.1.3. Quiénes participan en SET 77
4.1.4. Cómo funciona SET 78
4.1.5. Certificados digitales en SET 80
4.1.6. Ejemplos de fraudes 81
4.2. Especificación del protocolo SET 83
4.2.1. Fase de Compra 85
4.2.1.1. Recursos Homogéneos 89
4.2.1.2. Recursos Heterogéneos 92
4.2.2. Fase de Registro 94
4.2.2.1. Registro del Comprador 95
4.2.2.2. Registro del Vendedor 97
4.3. Análisis de prestaciones en SET 99

5. Aplicación: Sistemas de Fabricación Flexibles **107**
5.1.Qué es un Sistema de Fabricación Flexible 107
5.1.1. Concepto de Flexibilidad. Diferentes visiones 108
5.1.2. Beneficios/Costes de los FMS 111
5.2. Dónde se enmarca nuestro trabajo 113
5.3. Trabajos relacionados 120
5.4. Especificación de FMSs 122
5.4.1. Metodología 122

5.4.2. Caso de estudio 124
5.5. Evaluación de prestaciones en un FMS 129

6. Conclusiones y Trabajo Futuro **137**
6.1. Publicaciones 139
6.2. Trabajo futuro 140

Índice de cuadros

2.1. Semántica denotacional de CSP 21

2.2. Semántica operacional de CSP 22

2.3. Axiomas y reglas para procesos finitos 23

2.4. Axiomas y reglas para procesos recursivos 23

2.5. Axiomas para procesos derivados 24

3.1. Semántica Operacional: Prefijo y Prefijo Temporizado 42

3.2. Semántica Operacional: Elección Interna y Elección Externa . 42

3.3. Semántica Operacional: Paralelismo (Sincronización) 43

3.4. Semántica Operacional: Paralelismo 43

3.5. Semántica Operacional: Recursión 44

3.6. Suma de dos matrices con distintos tamaños y diferentes números de procesadores 51

3.7. Semántica Operacional con Recursos Heterogéneos 54

3.8. Semántica Operacional Definitiva: Prefijo y Prefijo Temporizado 64

3.9. Semántica Operacional Definitiva: Elección Interna y Externa 65

3.10. Semántica Operacional Definitiva: Sincronización 66

3.11. Semántica Operacional Definitiva: Paralelismo 66

3.12. Semántica Operacional Definitiva: Recursión 66

3.13. Semántica Operacional Definitiva: con recursos Heterogéneos, Expropiativos y no Expropiativos 67

4.1. Tiempo necesario para alcanzar el estado final con distinto valores en el número de compradores y vendedores 103

4.2. Evaluación de Prestaciones en un sistema con 5 Vendedores . . 106

5.1. Significados de Flexibilidad 108

5.2. Resultados para procesar 100 piezas en la Celda de ensamblaje 134

5.3. Resultados para Robot Lento en la Celda de Ensamblaje . . . 135

Índice de figuras

3.1. Grafo de Transiciones 46
3.2. Algoritmo de Evaluación de Prestaciones 47
3.3. Grafo de transiciones para la suma de dos matrices 3x3 con $\mathcal{N} = 2$ 50
3.4. Especificación Oficina de Correos con **BTC** 56
3.5. Especificación Oficina de Correos con la extensión temporal del álgebra CSP desarrollada por Roscoe 57
3.6. Especificación Taller Mecánico con **BTC** 59
3.7. Especificación Taller Pintura de piezas 62
3.8. Especificación del Comedor con Recursos expropiativos 68
3.9. Especificación del Comedor sin Recursos expropiativos 70

4.1. Protocolo de Petición de Compra 86
4.2. Protocolo de Autorización de Pago 87
4.3. Protocolo de Captura del pago 88
4.4. Especificación de la Fase de Compra del Protocolo SET con Recursos Homogéneos 92
4.5. Especificación de la Fase de Compra del Protocolo SET con Recursos Heterogéneos 95
4.6. Protocolo de Registro del Comprador 96
4.7. Especificación del Protocolo de Registro del Comprador 97
4.8. Protocolo de Registro del Vendedor 98

4.9. Especificación del Protocolo de Registro del Vendedor 98
4.10. Zoom del Grafo de Transiciones de la Fase de Compra del Protocolo SET con cinco procesos Compradores y Vendedores. 100
4.11. Resultados obtenidos a partir de la Tabla 4.1. 105

5.1. Sistema de Fabricación Flexible 116
5.2. Celda de Fabricación Flexible Discreta de Transformación . . . 123
5.3. Especificación Celda de Fabricación Flexible de la Figura 5.2 . 124
5.4. Celda de Fabricación Flexible a estudio 126
5.5. Especificación del FMS de la Figura 5.4 128
5.6. Celda Flexible de Ensamblaje 130
5.7. Especificación de Celda de Ensamblaje 131

Capítulo 1

Introducción

Tradicionalmente, los métodos formales se han usado como herramienta en la especificación, diseño y análisis funcional de los sistemas concurrentes. Durante el diseño, los aspectos cuantitativos generalmente eran postergados y sólo más adelante, en la fase de implementación comenzaban a ser considerados. Evidentemente, esta forma de diseñar puede conducir a la implementación de un sistema que funcionalmente cumple las especificaciones dadas, pero tal que sus características de funcionamiento (aspectos temporales, eficiencia, ...) lo hacen inservible. Esto puede hacer necesario un rediseño total del sistema, situación totalmente no deseable debido a los costes que introduce en el desarrollo del mismo.

Parece por tanto razonable que en la fase de diseño no tengamos sólo en cuenta los aspectos funcionales, sino que consideremos también los aspectos cuantitativos como nuevas restricciones a tener en cuenta. Una solución sencilla que nos permite introducir dichos aspectos cuantitativos en la fase de diseño es aumentar la expresividad del lenguaje de especificación de forma que dicha información pueda ser introducida de una forma clara y sencilla.

En esta tesis, hemos ampliado el poder expresivo del álgebra de procesos CSP. La hemos convertido en un álgebra de procesos temporizada llamada **BTC** capaz de tomar en consideración el número y tipo de recursos existentes en cada momento en un sistema. Esta ampliación trae consigo una mejoría significativa a la hora de especificar sistemas reales donde el uso de recursos suele ser de vital importancia. Algo menos evidente, y sin embargo también de gran importancia, es el hecho de que nos lleva a adentrarnos en temas pertenecientes a la teoría de scheduling, ya que cuando el recurso compartido que se está considerando es el procesador estamos hablando de planificación propiamente dicho.

Así, el objetivo de esta tesis es desarrollar un álgebra de procesos que pueda ser usada para especificar sistemas concurrentes que trabajen tanto con paralelismo real como con interleaving, en la que se pueda tener en cuenta que los recursos de que dispone un sistema deben de ser compartidos por los distintos procesos que en un momento determinado se encuentran en ejecución en dicho sistema. Esto es, suponemos que cualquier sistema está formado por un número de procesos que deben compartir uno o más recursos, cuando hablamos de recurso nos referimos a cualquier dispositivo que un proceso pueda necesitar en el transcurso de su ejecución, ya sea un procesador, un circuito hardware, un bus, un robot, un componente software o cualquier otra cosa. No importa que tipo de recurso sea, nuestra álgebra es capaz de modelar el uso de recursos heterogéneos, el requisito importante es que siempre que haya más procesos intentando usar un mismo tipo de recurso que recursos disponibles en el sistema, entonces no todos los procesos podrán ejecutarse a la vez (en el mismo momento). El ejemplo más común es un sistema multitarea en el que se comparte uno o más procesadores pero, como veremos en los casos de estudios presentados, puede aplicarse a sistemas mucho más complejos.

Además, habrá que tener en cuenta que los recursos de un sistema pueden ser de distintos tipos y que no siempre se pueden tratar igual (este es el caso de los recursos expropiativos y no expropiativos), por lo que habrá que tenerlo en consideración a la hora de diseñar el álgebra si queremos que sea capaz de modelar un mayor número de sistemas reales.

Como base de nuestra álgebra hemos optado por CSP por aunar sencillez y poder suficiente para expresar el paralelismo, el no determinismo y la sincronización. La hemos enriquecido ampliando y modificando su sintaxis y semántica operacional de manera que nos permita reflejar el tiempo de ejecución de cada acción, al mismo tiempo que se gestiona el paralelismo teniendo en cuenta el número de recursos disponibles en el sistema en cada momento. Con estas modificaciones conseguimos expresar el concepto de "consumo de recurso" que, como es bien sabido, cada día adquiere más importancia al ir casi siempre ligado a cuestiones económicas.

Una vez presentada nuestra nueva álgebra **BTC**, nos centramos en el análisis de prestaciones del sistema, concretamente y tratándose de una álgebra temporal, nos centramos en el análisis temporal. Para ello, a partir de la semántica operacional, construimos para el sistema un grafo de transiciones (un grafo dirigido con pesos) del que abstraemos la información sobre las acciones para centrarnos en la información temporal.

Una vez obtenido el grafo, lo que pretendemos resolver es un problema de maximizar el rendimiento relativo a minimizar el tiempo necesario para alcanzar el estado final (a partir del inicial). Para ello debemos encontrar el camino más corto (con menos peso) entre estos dos estados y lo hacemos definiendo un algoritmo, modificación del algoritmo de Dijkstra, que nos permita calcular el tiempo necesario para evolucionar entre estados.

Después de hacer diversas comprobaciones sobre la adecuación de las especificaciones obtenidas con **BTC** a los sistemas que intentamos modelar y de un estudio detallado de los resultados obtenidos con nuestro análisis de prestaciones temporales sobre pequeños sistemas de prueba, pasamos a enfrentarnos con sistemas reales.

Como sistemas reales a estudiar hemos optado por temas que tengan un fuerte impacto actualmente. Uno de ellos es indudablemente Internet que en la última década se ha convertido en una de las tecnologías más dominantes en el campo de los computadores. Dentro de la revolución general causada por su aparición, ha surgido una gama de nuevas tecnologías y de entre ellas hemos seleccionado una que nos ha parecido de gran interés por estar en continuo desarrollo: El Comercio Electrónico, el cual ha permitido redefinir la manera en que se hacían negocios tradicionalmente.

La mayoría de las transacciones llevadas a cabo a través del Comercio Electrónico conlleva un tráfico de dinero de una manera u otra, el modo más común es utilizar tarjetas de crédito, pero sea cual sea el modo utilizado, lo cierto es que el pago debe de hacerse a través de una red que por definición es no segura lo que ha traído consigo la aparición de protocolos de seguridad. De entre estos protocolos cabe destacar el protocolo SET ya que, a diferencia del resto de protocolos que son de carácter general (seguridad en general), SET fue expresamente diseñado para el comercio electrónico. Evidentemente, todos ellos necesitan ser estudiados a un nivel formal y, de hecho, así se está haciendo, pero en el caso concreto del protocolo SET encontramos que han aparecido bastantes dificultades al ser el más complejo en diseño y completo en prestaciones. Así que nos ha parecido un reto interesante y, en esta tesis, nos hemos lanzado a realizar su especificación y estudiar sus prestaciones temporales.

Como segundo sistema real a estudiar hemos escogido un Sistema de Fabricación Flexible debido principalmente a dos razones; la primera de ellas es por que pertenece a un campo en el que los avances tecnológicos se suceden a gran velocidad y, evidentemente, se necesita disponer de una herramienta con la que poder decidir la manera más adecuada para adaptarse a ellos. Por ejemplo, poder hacer comparaciones en las prestaciones de distintas confi-

guraciones de un sistema antes de adoptarlo es, sin lugar a duda, una gran ayuda a la vez que un ahorro de tiempo y por consiguiente económico.

La segunda razón por la que se ha elegido los Sistemas de Fabricación Flexible es por la adaptación tan precisa de **BTC** a este tipo de sistemas. En ellos el impacto en el rendimiento total del sistema debido a una buena utilización de los recursos es notoria y, además, son sistemas que precisamente se caracterizan por su flexibilidad debido a la necesidad de adaptarse constantemente al entorno, cada vez más competitivo, en el que trabajan ya que ahora los productos tienen un ciclo de vida mucho más corto y las necesidades de los clientes son más variables.

En esta tesis detallaremos tanto estos casos de estudio como el soporte teórico que hemos desarrollado y presentaremos un estudio detallado de los trabajos similares existentes con el fin de constatar que a pesar de que todos ellos, en menor o mayor medida, presentar mejorías respecto a versiones anteriores, realmente ninguno abarca todo lo que **BTC** es capaz.

Resumen de la tesis

El contenido del resto de capítulos de esta tesis es el que se indica a continuación:

Capítulo 2: Lenguajes formales de especificación de sistemas concurrentes.

En este capítulo presentaremos los formalismos en los que nos basaremos a lo largo de esta tesis para desarrollar nuestro lenguaje de especificación y representar los sistemas que trataremos. Nos centraremos en la representación intuitiva de los principales conceptos que subyacen a las *álgebras de procesos* y especialmente en el álgebra de procesos CSP ya que será la que hemos tomado como referencia para desarrollar **BTC**.

El objetivo aquí no será el de presentar una larga introducción al estado del arte dentro del área de los lenguajes de especificación de sistemas concurrentes que sin lugar a duda es cada día más extensa, sí no que lo que intentaremos será presentar los conceptos esenciales presentes en dichos lenguajes de una manera lo más amena posible, en especial en lo referente a aquellos aspectos que utilizaremos posteriormente a lo largo de esta tesis.

Capítulo 3: Álgebra de procesos temporizada BTC.

En este capítulo presentamos el álgebra de procesos que hemos desarrollado y que es parte fundamental de esta tesis: **BTC**. Empezaremos justificando la necesidad de desarrollar un álgebra de procesos nueva. Esto lo haremos examinando y comparando los modelos ya existentes los cuales, según nuestro criterio, no sólo no cumplen todos los requisitos que **BTC** sino que además en alguno de ellos se pueden encontrar algunos puntos cuestionables.

A continuación presentaremos **BTC** por medio de su sintaxis y su semántica operacional. Esto lo haremos de una manera "incremental", quiere decir, hemos pensado que la manera más intuitiva de entender nuestro lenguaje sería presentar todas sus características poco a poco. Empezamos mostrando una primera versión capaz de tener en cuenta tanto la duración de las acciones como el entorno en el que se ejecutan, esto es, el número de recursos existentes en el sistema. Esta versión, aún siendo la primera, no es ni mucho menos trivial, ya que nos permitirá mostrar en detalle el hecho de que **BTC** es capaz de trabajar tanto en interleaving como en paralelismo real uniendo conceptos de álgebras de procesos con los propios de la teoría de scheduling.

Una segunda versión mostrará los cambios efectuados tanto en la sintaxis como en la semántica operacional para poder tratar con distintos tipos de procesos al mismo tiempo (recursos heterogéneos) incrementando así de una manera considerable la potencia de **BTC**.

En la tercera y última versión que presentaremos clasificamos los recursos en dos tipos: expropiativos y no expropiativos. Esto se hace necesario debido a que a pesar de nuestra negativa a adoptar la idea usada habitualmente en la teoría de scheduling de asignación estática de recursos, hemos descubierto que algunos recursos (los no expropiables) así lo requieren. Por consiguiente, seguiremos teniendo asignación de recursos dinámica para los que sean expropiativos y utilizaremos la estática para los no expropiativos.

Todo esto lo haremos para presentar el álgebra de procesos que hemos desarrollado con el fin de hacer las especificaciones de los sistemas a tratar de la manera más fiable y recogiendo, al mismo tiempo, tanta información como sea posible. Pero además, en este capítulo realizaremos evaluación de prestaciones. Para ello presentamos la metodología que hemos desarrollado haciendo uso de una variante del algoritmo de Dijkstra con el fin de poder realizar análisis de rendimiento (a nivel temporal) del sistema.

A lo largo de todo el capítulo hemos incluido diversos ejemplos con el fin facilitar la lectura y proporcionar un modo de comprensión más ameno. También hemos utilizado un sencillo ejemplo para mostrar las mejoras más evidentes que proporciona nuestro lenguaje comparándola con un álgebra de procesos temporal bien conocida: la desarrollada por Roscoe.

Capítulo 4: Protocolo SET.

Este capítulo comenzará explicando el por qué se ha elegido el protocolo SET como caso de estudio. Continuará con una visión sobre qué es y para qué sirve este protocolo antes de pasar a explicar, de la manera más intuitiva posible, como funcionan los diversos protocolos que forman SET (trabajo no trivial tratándose del protocolo de seguridad más complejo presente actualmente en el mercado).

Una vez explicado qué se quiere modelar pasamos a realizar su especificación para lo que debemos modelar los distintos protocolos de las dos fases que constituyen el protocolo SET. Primero realizamos la especificación utilizando sólo recursos homogéneos con el fin de obtener una visión clara y más tarde se muestra la especificación con recursos heterogéneos para tener una especificación más próxima al sistema real.

También se realizará un análisis del rendimiento de este sistema del que obtendremos conclusiones interesantes.

Capítulo 5: Sistemas de Fabricación Flexible.

Modelar Sistemas de Fabricación Flexibles es un reto apasionante pero no exento de gran complejidad ya que dichos sistemas reúnan todos los elementos que potencialmente pueden estar presentes en los sistemas concurrentes; son sistemas guiados por eventos, con acciones que deben de suceder de modo asíncrono, relaciones secuenciales y concurrentes, donde existe no determinismo y pueden aparecer conflictos que pueden desembocar en bloqueos y por supuesto cuentan con recursos que necesitan ser utilizados en exclusión mutua. Pero también debido a esta complejidad se hace más imprescindible el uso de métodos formales para validar dichos sistemas.

Nosotros, una vez situado nuestro trabajo dentro del amplio campo que constituyen los Sistemas de Fabricación Flexible, hemos especificado dos sistemas que hemos elegido como caso de estudio y hemos realizado análisis de sus rendimiento con distintas configuraciones. Estos estudios nos han ayudado a demostrar lo importante que es poder

disponer de una herramienta como **BTC** que permite poder hacer estudios comparativos de los sistemas propuestos antes de ser implantados.

Capítulo 6: Conclusiones y Trabajo Futuro.

En este último capítulo de nuestra tesis presentaremos las conclusiones y las distintas líneas por las que puede continuar nuestro trabajo.

Capítulo 2

Lenguajes Formales de Especificación de Sistemas Concurrentes

En este capítulo presentaremos los formalismos en los que nos basaremos a lo largo de esta tesis para desarrollar nuestro lenguaje de especificación y representar los sistemas que trataremos. Nos centraremos en la representación intuitiva de los principales conceptos que subyacen a las *álgebras de procesos* [Hoa78, Hoa85, Mil89, BW90, Mil99, BPS01]. Aunque para tener una visión algo más global lo haremos de una manera más general y también veremos otras nociones para representar sistemas concurrentes como son los *sistemas etiquetados de transiciones* y las *máquinas de estados finitos*. Todos estos formalismos comparten la característica principal de ser (o servir de) *lenguajes de especificación*. Ello significa que, en general, la utilidad de estos lenguajes no residirá en dotar de una sintaxis y de una semántica bajo las cuales se realicen programas que resuelvan nuestros problemas computacionales *finales*, sino que facilitarán un marco en el cual estudiar *otros programas* o *sistemas* que resuelven tales problemas computacionales. Es por ello que podemos considerarlos lenguajes de segundo orden o *metalenguajes*. Es más, el propio nombre de *especificación* nos indica que gracias a dichos lenguajes el análisis de los sistemas se podrá realizar con anterioridad a la *implementación* del mismo. De hecho, tal análisis podrá ser independiente de la realización práctica del sistema correspondiente. Por otra parte, los lenguajes de especificación también permitirán realizar análisis basados en la comparación del comportamiento del sistema final obtenido con el comportamiento previsto por la semántica del lenguaje de especificación para el modelo correspondiente.

El objetivo de este capítulo no será el de presentar una larga introducción al estado del arte dentro del área de los lenguajes de especificación de sistemas concurrentes que sin lugar a duda es cada día más extensa, sí no que lo que intentaremos será presentar los conceptos esenciales presentes en dichos lenguajes de una manera lo más amena posible para el lector, en especial en lo referente a aquellos aspectos que utilizaremos posteriormente a lo largo de esta tesis.

2.1. Objetivos de los lenguajes formales de especificación

El principal objetivo de los lenguajes formales de especificación es el de servir de apoyo para el estudio de otros sistemas computacionales (implementados o no).

La aplicación de los lenguajes de especificación para el estudio de sistemas se puede dividir en dos pasos. El primero consiste en la representación de las características *relevantes* del sistema a analizar utilizando la sintaxis del lenguaje de especificación correspondiente. Cabe destacar que los lenguajes de especificación son, en general, de más alto nivel que los lenguajes en los que los sistemas a analizar están/serán implementados. Por ello, algunos de los aspectos concretos de la implementación serán usualmente ignorados. Tal abstracción de las características relevantes será la razón de la potencia de análisis que nos proveerán estos lenguajes: a través de la eliminación de algunos aspectos irrelevantes, el análisis automático o sistemático de las propiedades relevantes del sistema se hará (parcial o totalmente) posible.

El segundo paso de la aplicación de los lenguajes de especificación consiste en la manipulación de la especificación formal del sistema de forma que se consigan extraer las propiedades relevantes de dicho sistema. Dicha manipulación se apoya, en general, en el amplio conocimiento que se dispone de la semántica del lenguaje de especificación. Esta permite establecer de antemano aspectos tales como las posibles evoluciones futuras de dicho sistema, el cumplimiento de ciertos invariantes, o la aparición de ciertas propiedades indeseables. De esta forma, el análisis del sistema puede consistir bien en el estudio de las propiedades de su especificación o bien en una cierta comparación entre la especificación y una implementación dada que, supuestamente, es *conforme* con la especificación.

Nótese que ninguno de los dos pasos de la aplicación de los lenguajes formales al análisis de sistemas será, en general, completamente automatizable. Dependiendo de las características consideradas como relevantes en el primer paso, o de las propiedades a analizar en el segundo, el desarrollo de procedimientos automáticos de análisis que resulten correctos y/o completos en su estudio de propiedades será o no posible. No obstante, incluso en el caso de que alguno de los pasos no pueda llevarse a cabo de manera automática, el poder de análisis de dichas técnicas las convierte en herramientas imprescindibles para el estudio formal de sistemas computacionales de diversa índole.

La potencia de análisis de los lenguajes de especificación se basa en su capacidad para extraer las características relevantes de los sistemas a modelar. Por lo tanto, se deberán fijar en primer lugar las características que se consideran relevantes para el análisis concreto que desee realizarse. El principal objetivo en el que nos centraremos en este capítulo es el de analizar el comportamiento de sistemas *concurrentes o distribuidos*. La característica más relevante de dichos modelos es aquella intrínseca a tal tipo de sistemas: la *comunicación entre procesos*, elemento indispensable para el funcionamiento de cualquier sistema concurrente.

En consecuencia, los lenguajes de especificación centrados en la descripción de sistemas concurrentes tendrán como objetivo el modelado detallado de la comunicación entre los procesos presentes en el sistema. En esta línea se intentarán abstraer, en mayor o menor medida, todos los demás aspectos de tales sistemas. Ello implica que el comportamiento *interno* de cada proceso, es decir, el conjunto de acciones ejecutadas por el proceso que no suponen una comunicación con otros procesos, se abstraerá de forma que no estará representado en los modelos.

Sin embargo, esta limitación afectará a la interpretación de aspectos que sí deseamos representar, es decir, a las comunicaciones entre los procesos. Ello es así porque las acciones desarrolladas internamente por los procesos determinarán qué comunicaciones deben llevarse a cabo con otros procesos. Por lo tanto, inevitablemente, la abstracción de ciertos detalles que no son *directamente* interesantes hará que, *indirectamente*, perdamos cierta información relevante sobre aquellos aspectos que consideramos interesantes.

Vemos por tanto que todo lenguaje de especificación *centrado* en el estudio de las comunicaciones entre procesos en los sistemas concurrentes convertirá, en virtud de la abstracción que realiza, ciertas *certidumbres* acerca del comportamiento que modela en *incertidumbres*. Concretamente, las *elecciones*

que se produzcan de manera interna en el sistema relativas a realizar una determinada acción de comunicación u otra dejarán de ser predecibles en el modelo resultante, apareciendo así un cierto *indeterminismo*.

2.2. Ingredientes de los lenguajes

En esta sección estudiaremos los elementos con los que deben contar los lenguajes de especificación de sistemas concurrentes. A pesar de las diferencias existentes entre los distintos tipos de lenguajes, los conceptos fundamentales que consideran son muy similares, por lo que el número de ingredientes básicos comunes a todos ellos es alto. Ello permite que la descripción desarrollada en esta sección sea general, si bien en ocasiones particularizaremos algunos de los conceptos vistos para cada lenguaje específico.

En general, todo lenguaje cuenta al menos con una *sintaxis* y una *semántica operacional*. La primera nos da el conjunto de expresiones que forman parte del lenguaje y que, por tanto, son susceptibles de recibir un significado concreto. Por su parte, la semántica operacional nos da el conjunto de reglas que determinan la manera en que las expresiones de nuestro lenguaje evolucionan hacia otras expresiones. En el caso concreto de los lenguajes de especificación de sistemas concurrentes, dicha evolución debe interpretarse como el cambio de estado debido a la ejecución del sistema. Es decir, la aplicación de una regla de la semántica operacional representa la evolución del sistema en el tiempo.[1]

Vemos por tanto, que el *comportamiento* de un sistema estará representado por las distintas evoluciones que se puedan producir desde el estado inicial de dicho sistema como resultado de la aplicación de las reglas dadas en la semántica operacional.

La clave de este tipo de lenguajes consistirá en lograr que comportamientos complejos globales surjan a partir de los comportamientos simples de cada una de las partes, por lo que parece razonable por tanto que la definición de la semántica operacional de tales lenguajes esté *estratificada* de una forma

[1]Sin embargo, no toda aplicación de una regla de la semántica operacional conllevará siempre el paso del *tiempo*. Tanto en los lenguajes que representan explícitamente el tiempo como en los que no lo hacen, algunas transiciones podrán representar acciones *instantáneas*, es decir, acciones que no consumen tiempo para realizarse. No obstante, en general será una característica indeseable el que dichas acciones puedan concatenarse indefinidamente pues, en tal caso, la aplicación de la semántica operacional no significaría *globalmente* siempre una evolución hacia el futuro.

similar. Es decir, la semántica operacional deberá ser capaz de describir el comportamiento de cada una de las partes de un sistema y, a partir de dichos comportamientos individuales, proporcionar el comportamiento global del sistema. Para que dicha descripción jerárquica de la semántica operacional sea posible desearemos, en general, que el comportamiento de cada *parte* pueda definirse sin hacer referencia al comportamiento del *todo*. De esta forma, el comportamiento de cada nivel de distinta complejidad jerárquica podrá calcularse a partir del comportamiento de los niveles inferiores. Diremos entonces que dicha semántica operacional es *composicional*.

Por ejemplo, en el caso de las álgebras de procesos las reglas de una semántica operacional se suelen presentar en forma de premisas y conclusión [Plo81] orientadas a sintaxis; cuando las partes cumplen lo indicado en las premisas (es decir, si las partes pueden evolucionar de una determinada manera indicada) para el todo se tiene el comportamiento definido por la conclusión (es decir, cierta evolución global se puede realizar).

Veamos ahora el tipo de comportamientos básicos que desearemos poder expresar en un lenguaje de especificación de sistemas concurrentes a través de su semántica operacional. Denominaremos *acción* a la manifestación más básica de ejecución en el sistema. Con el objetivo de mantener un conjunto de acciones básicas manejable, en ocasiones se agrupan varios eventos de comunicación similares en una sola acción que las representa a todas, si bien tal acción puede ser *etiquetada* con uno o varios *parámetros* que permiten identificar el evento de comunicación exacto que representa.

Otra clasificación habitual entre las acciones es la distinción entre acciones de *entrada* y acciones de *salida* dependiendo del *papel* de la acción dentro de cada elemento del sistema. Si bien la diferenciación entre acciones de entrada y salida no es necesaria en las álgebras de procesos, esta sí se da habitualmente en las máquinas de estados finitos y en los sistemas etiquetados de transiciones. La diferencia entre ambos formalismos radica en que mientras que en las máquinas de estados finitos las acciones de entrada y las acciones de salida van emparejadas en una misma transición, cada transición de un sistema etiquetado de transiciones va etiquetada bien por una acción de entrada, bien por una acción de salida. Como veremos más adelante, en algunos contextos los lenguajes de especificación podrán considerar un tercer tipo de acciones: las acciones *internas* u *ocultas*.

Una vez que podemos expresar los eventos básicos de comunicación, necesitaremos una manera de expresar la *secuenciación* de dichos sucesos, es decir, el hecho de que *después* de unas acciones se producen otras. Dicha secuenciación de sucesos es relativa a cada proceso. Es decir, dentro de cada

proceso la acción inmediatamente posterior a una dada será la que delimite la secuenciación de sucesos de dichos procesos. Por tanto, los constructores sintácticos del lenguaje nos deberán permitir expresar el hecho de que después de una determinada acción sucede otra acción determinada. En el caso de las álgebras de procesos, tal suceso tendrá su propio constructor sintáctico (por ejemplo ;), mientras que en el caso de las máquinas de estados finitos y los sistemas etiquetados de transiciones tal suceso surge implícitamente a través de la aplicación consecutiva de transiciones entre estados.

También necesitaremos una manera de expresar el *indeterminismo* inherente a cualquier modelo creado con dichos lenguajes. Es decir, necesitaremos contar con alguna manera de expresar el hecho de que el sistema podría tomar caminos distintos dependiendo de factores o sucesos que no podemos conocer de antemano en el momento de construcción del modelo. Encontramos indeterminismos en dos niveles diferentes del sistema. Tanto las elecciones que se produzcan de manera interna en el sistema relativas a la ejecución de una determinada acción u otra como las tomadas por influencia de una interacción con el entorno externo (que no podemos controlar), representarán un cierto tipo de indeterminismo que se da *a nivel de proceso*. Por otro lado, el indeterminismo debido a la libertad en la secuenciación de las acciones concurrentes de los distintos procesos surge como resultado de la evolución conjunta de los procesos, por lo que dicho indeterminismo surge *a nivel de sistema*.

Como dijimos anteriomente, los constructores sintácticos proporcionados por los lenguajes de especificación deberán favorecer la composicionalidad de la semántica operacional asociada. Es por ello que, en general, será recomendable que el indeterminismo a nivel de proceso y el indeterminismo a nivel de sistema se expresen por medio de constructores sintácticos diferentes.

El indeterminismo a nivel de proceso recoge el indeterminismo debido a todos los factores no predecibles que tienen una repercusión directa en el comportamiento de un proceso *independientemente* de su relación con los demás procesos del sistema. En los modelos algebraicos clásicos, esta situación se describe con uno (o varios) constructores sintácticos, denominados *operadores de elección* (por ejemplo +). En las máquinas de estados finitos y en los sistemas etiquetados de transiciones dicha situación se expresa por medio de varias transiciones saliendo de un mismo estado. A pesar de tener una representación similar, en general será posible distinguir entre el indeterminismo debido al entorno desconocido y el indeterminismo debido a decisiones cuyos motivos no están representados. Así, la propia distinción entre acciones de entrada y acciones de salida puede ayudar a hacer tal distinción.

También es posible enriquecer el lenguaje con un tipo de acciones especiales que denoten explícitamente dichas decisiones internas. Tal es el caso de las *acciones internas* las cuales abstraen una decisión interna del proceso cuyas causas no están representadas en el modelo. La consideración de dichas decisiones como *acciones* no será pues más que una manera compacta de representar tales decisiones de forma que se ajusten a las formas sintácticas ya existentes. Habitualmente se utiliza un único símbolo para representar esta actividad interna, pues suele ser irrelevante distinguir entre acciones internas de diversos tipos: todas ellas representan decisiones tomadas internamente en base a motivos que deseamos ignorar. En las álgebras de procesos, tales acciones especiales se representan habitualmente por medio de una τ.

El indeterminismo a nivel de sistema recoge el indeterminismo debido a todos los factores no predecibles que tienen una repercusión directa en el sistema debido a la relación de los distintos procesos entre sí. Este indeterminismo representa, por ejemplo, la libertad de los procesos concurrentes de ejecutarse de manera asíncrona. Así, un modelo podría representar el hecho de que dos procesos dispuestos a ejecutar respectivamente a y b podrían evolucionar de forma que el sistema globalmente mostrara una a y después una b, o bien mostrara una b y después una a, *siempre y cuando* el modelo desee proporcionar tal libertad a los procesos en dicho momento concreto. De esta manera, si deseáramos que la acción a del primer proceso se sincronizará con la acción b del segundo, entonces ambas acciones tendrían que producirse necesariamente a la vez y dicha libertad desaparecería. El indeterminismo a nivel de sistema representa también la incertidumbre producida por el hecho de que se permita a dos procesos sincronizar por medio de dos pares de acciones *diferentes* en una misma situación dada.

El indeterminismo a nivel de sistema se representa en las álgebras de procesos por medio de uno (o varios) constructores sintácticos específicos, denominados en general *operadores de paralelo* (por ejemplo $\|$). En este caso el mismo operador rige tanto el indeterminismo debido a la evolución asíncrona de los procesos como la evolución síncrona de los mismos (debida habitualmente a las comunicaciones entre ellos).

En el caso de las máquinas de estados finitos o de los sistemas etiquetados de transiciones, o bien se proveen constructores sintácticos específicos o bien se aplican reglas de composición que permitan obtener la máquina representativa de un sistema a partir de las máquinas de los correspondientes procesos, de forma que dicho indeterminismo quede representado en la máquina compuesta.

Además de los tipos de operaciones vistos anteriormente, algunos lenguajes proveen operadores adicionales para aumentar la complejidad de los escenarios que se pueden describir con ellos. Tal es el caso de los operadores de *recursión* en las álgebras de procesos, que permiten definir los procesos en función de sí mismos, lo que les confiere la capacidad de denotar patrones de comportamiento repetitivos. En el caso de las máquinas de estados finitos y los sistemas etiquetados de transiciones, tales patrones de comportamiento se pueden lograr *parcialmente* por medio de la introducción de ciclos. Sin embargo, en el caso de las álgebras la recursión facilita una mayor expresividad pues permite, por ejemplo, expresar un proceso como un sistema formado por otro proceso y él mismo. Esto permite la creación de nuevos niveles jerárquicos (potencialmente infinitos) como resultado de la evolución del sistema. De esta forma, la propia *descripción* del sistema cambia a medida que el sistema evoluciona, hasta el punto de que son concebibles infinitas formas futuras posibles a partir de una dada. Por contra, el número de configuraciones posibles en una máquina de estados finita a partir de una configuración inicial es finita, lo que le da una menor expresividad (en concreto, la de los lenguajes regulares).

2.3. Álgebra de procesos CSP

CSP es un lenguaje algebraico utilizado para describir sistemas compuestos por procesos que evolucionan de forma concurrente. Fue introducido por Hoare en 1978 [Hoa78]. Posteriormente, el mismo autor publicó un texto sobre CSP [Hoa85], donde se estudia la semántica del modelo. La idea básica en que se apoya CSP consiste en considerar que los procesos evolucionan de una forma independiente, hasta que deban ejecutar acciones comunes a varios de ellos, para lo cual habrán de sincronizar. Este mecanismo de sincronización permite además modelar la comunicación entre procesos. En el modelo, como sucede en general en todos aquellos campos en los que se aplican técnicas de abstracción, se ignoran todas aquellas características que resultan irrelevantes para el estudio de la cuestión en la que hemos fijado nuestro interés, que en este caso es la *concurrencia*. En consecuencia se abstrae el significado concreto de las acciones ejecutadas por los procesos, que en consecuencia quedan exclusivamente representadas por el identificador que las denota.

La sintaxis del lenguaje CSP viene dada por una signatura sobre un único género *de procesos*, que incluye operadores para representar todos los elementos fundamentales de la concurrencia, unos explícitamente, y otros por medio de combinaciones de dichas operaciones. El punto de partida sobre

el que se definen los procesos de CSP es el conjunto de acciones que los mismos pueden ejecutar y que representan la realización de eventos por el mismo. Como ya hemos precisado, en CSP no nos importa el significado de las acciones; éstas pueden representar intuitivamente cuestiones tan diversas como asignaciones en un lenguaje de programación, la introducción de una moneda en la ranura de una máquina, el funcionamiento de un reloj, el envío de una señal telefónica, etc.

El conjunto de acciones que un proceso puede realizar se denomina su *alfabeto*. Dentro del mecanismo de abstracción de las características de las acciones que se consideran irrelevantes para nuestro estudio, entendemos que las acciones no llevan asociada una duración concreta, si bien asumimos que la misma es siempre finita. Dicha abstracción nos lleva a considerar la ejecución de las acciones como un proceso instantáneo. En consecuencia, el factor tiempo, en lo que de cuantitativo tiene, no será tenido en cuenta al realizar la descripción de procesos mediante este lenguaje (en esta primera versión), ni por tanto podrá ser estudiado en el marco del modelo. Ello por supuesto no implica que el tiempo no se refleje en términos cualitativos, pues como veremos es precisamente la ordenación temporal (relativa) de las acciones ejecutadas por un proceso, la que nos proporciona la semántica del mismo.

Otra consecuencia de la visión de un proceso como un agente capaz de ejecutar una serie de acciones (totalmente) ordenadas temporalmente es que en CSP, al menos en tanto nos quedemos con su semántica de entrelazamiento (*interleaving*) clásica, no es posible describir la ejecución simultánea de acciones, o más exactamente la necesidad o imposibilidad de que tal cosa suceda. Así, cuando no exista relación de causalidad entre dos acciones, tendremos la posibilidad de ejecutar las mismas en un orden cualquiera.

Podríamos en cambio considerar que cuando aparece la necesidad de sincronizar para ejecutar una cierta acción, justamente se está imponiendo la ejecución simultánea de las distintas acciones que sincronizan. Ello sería admisible desde el punto de vista intuitivo, pero no así a nivel formal, pues cuando se produce la ejecución de una acción por medio de una cierta sincronización, se entiende que dicha acción es una sola, si bien de cara a su ejecución deben colaborar los distintos procesos que se sincronizan.

Como convenio de notaciones utilizaremos para denotar las acciones ejecutadas las primeras letras minúsculas: $a, b, \ldots,$; los conjuntos de acciones se

denotarán con las primeras letras mayúsculas: $A, B, \ldots,$ etc. Por último, para denotar los procesos se usarán las siguientes letras mayúsculas: $P, Q, \ldots, .$

Operadores del lenguaje

Describiremos brevemente las operaciones que forman la signatura del álgebra de procesos CSP ya que más tarde, al presentar nuestra álgebra de procesos **BTC** algunos de ellos permanecerán iguales. No entraremos sin embargo, en un análisis de las propiedades de las mismas, cuyo desarrollo detallado puede encontrarse en [Hoa85].

La sintaxis de los procesos finitos del lenguaje CSP viene definida por la siguiente expresión BNF:

$$P \ ::= \ STOP \mid CHAOS \mid a \rightarrow P \mid P \sqcap P \mid P \,\Box\, P \mid P \parallel P \mid PQ \mid P \backslash a$$

y el significado intuitivo de cada uno de estos operadores es el siguiente:

STOP: El proceso más sencillo imaginable es el proceso que nunca hace nada. Se denota por STOP y simboliza una máquina incapaz de realizar acción alguna.

CHAOS: Un proceso que puede comportarse en todo momento de cualquier forma, siendo su comportamiento absolutamente imprevisible. Se trata en consecuencia del proceso más no determinista de todos.

PREFIJO: Nos permite indicar que un proceso comenzará ejecutando una determinada acción. En concreto, $a \rightarrow P$ realizará la acción a, y después pasará a comportarse de la forma indicada por el proceso P.

ELECCIÓN INTERNA: Nos permite definir procesos cuyo comportamiento es decidido por el sistema, escogiendo para ello entre sus dos argumentos a su libre arbitrio, sin intervención posible del medio exterior en dicha decisión, ni conocimiento (en principio) por parte del mismo de la decision tomada.

ELECCIÓN EXTERNA: El proceso $P \,\Box\, Q$ se comporta como P (resp. Q) si el entorno pide la realización de una acción que sólo puede hacer P (resp. Q). Si el entorno pide una acción que pueden hacer ambos, entonces es de nuevo el sistema quien toma libremente la decisión de cuál de los dos procesos será el que la ejecute, siendo el proceso escogido el que continuará ejecutándose.

CONCURRENCIA: El operador $\|$ permite agrupar procesos que se ejecutan concurrentemente, sincronizando para la ejecución de cada acción. Así, dada una acción a, cuando uno de los argumentos desea ejecutar dicha acción, debe esperar a que el otro se encuentre también en disposición de ejecutar esa misma acción. Por ejemplo, en una máquina de expedición de chicles, la acción moneda puede corresponder a la introducción de una moneda por parte de un usuario. Por tanto, cuando la máquina llega a ese punto, debe quedar esperando a que el usuario introduzca una moneda (es decir, ejecute también esa acción).

ENTRELAZADO: El operador $|||$ modela otro tipo de ejecución en paralelo en el cual no existe sincronización. En esta composición cada proceso puede evolucionar en respuesta a los estímulos del medio exterior con independencia del otro.

Estos dos últimos son los operadores paralelos de la versión original [Hoa78] de CSP, discutida en esta breve introducción al lenguaje. Sin embargo posteriormente en [Bro83] se ha desarrollado una nueva versión del lenguaje, en la que dichos operadores se sustituyen por un único operador de concurrrencia más general, el cual lleva un argumento adicional, que es un conjunto de acciones, que define el *conjunto de acciones de sincronización*. Así, el proceso $P \|_A Q$ corresponde a la ejecución concurrente de los procesos P y Q, los cuales deben sincronizar para ejecutar las acciones del conjunto A, mientras pueden ejecutar independientemente las acciones que no pertenezcan a dicho conjunto. Dado que esta nueva versión del modelo es más compacta, al contar con un operador menos, pero al mismo tiempo más flexible, dada la mayor generalidad del nuevo operador paralelo, se ha hecho práctica común la utilización de la misma.

OCULTACIÓN (HIDING): En un proceso puede interesarnos abstraer los efectos de alguna acción en particular, para lo cual dispondremos del operador de *ocultación*, cuyo efecto consiste en hacer invisibles, y por tanto incontrolables por el medio exterior, las ejecuciones de la acción ocultada.

De cara a la definición de procesos infinitos, que resultan de capital importancia cuando se desea modelar sistemas *reactivos*, el lenguaje cuenta con un operador recursivo adicional.

RECURSION: En CSP se utiliza la técnica descrita en la sección anterior, de introducción de procesos variables junto con los operadores (sintácticos) de punto fijo, para definir los procesos infinitos (recursivos). Así, $\mu X.(a \rightarrow X)$ representa un proceso que realiza indefinidamente la acción a, lo que ilus-

tra la interpretación habitual de un proceso $\mu X.P$ definido recursivamente, consistente en verlo como una solución (distinguida) de la ecuación asociada $X = P$.

Semántica denotacional de CSP

El comportamiento de un proceso lo describimos de manera denotacional por medio de su conjunto de *rechazos*. Un rechazo de un proceso es un par formado por una secuencia de acciones ejecutadas por el mismo (una *traza*) y un conjunto finito de acciones, cuyo significado es que el proceso puede ejecutar dicha traza, tras lo cual puede rechazar simultáneamente la ejecución de las acciones indicadas en el conjunto. Por tanto el conjunto de rechazos F que define un proceso es un subconjunto de $\mathcal{P}(\Sigma^* \times \mathcal{P}(\Sigma))$, que ha de satisfacer una serie de condiciones:

1. $(\langle\rangle, \emptyset) \in F$, donde $\langle\rangle$ representa la traza vacía.
2. $(st, \emptyset) \in F \Rightarrow (s, \emptyset) \in F$.
3. $V \subseteq W$ y$(s, W) \in F \Rightarrow (s, V) \in F$.
4. Sea $U = \{a \mid (sa, \emptyset) \in F\}$ y sea W un subconjunto finito de $\Sigma - U$, entonces $(s, V) \in F \Rightarrow (s, V \cup W) \in F$.

La función semántica $[\![\ ]\!]$ que define la semántica denotacional de CSP se define, como es habitual, por inducción estructural como se muestra en la Tabla 2.1.

Semántica operacional de CSP

Como hemos indicado, el significado operacional de un proceso vendrá dado por las posibles evoluciones del mismo. En este caso introducimos al efecto un par de sistemas de transiciones: uno etiquetado que representa las evoluciones con acciones visibles, y otro no etiquetado para indicar las evoluciones (internas) autónomas de un proceso. Así, la transición

$$P \xrightarrow{a} Q$$

representa que el proceso P ejecuta la acción a, y pasa a comportarse como el proceso Q, mientras que

$$P \longrightarrow Q$$

$$
\begin{array}{lcl}
[\![\text{STOP}]\!] & = & \{(\langle\rangle, X) \ / \ X \subseteq \Sigma, \quad X \text{ finito}\} \\
[\![\text{CHAOS}]\!] & = & \{(s, X)/s \in \Sigma^* \wedge X \subseteq \Sigma, \ X \text{ finito}\} \\
[\![a \rightarrow P]\!] & = & \{(\langle\rangle, X)/X \subseteq (\Sigma - \{a\}) \wedge X \text{ finito}\} \cup \\
 & & \{(\langle a\rangle s, X)/(s, X) \in [\![P]\!]\} \\
[\![P \sqcap Q]\!] & = & [\![P]\!] \cup [\![Q]\!] \\
[\![P \,\Box\, Q]\!] & = & \{(\langle\rangle, X)/(\langle\rangle, X) \in [\![P]\!] \cap [\![Q]\!]\} \cup \\
 & & \{(s, X)/s \neq \langle\rangle \wedge ((s, X) \in [\![P]\!] \cup [\![Q]\!])\} \\
[\![P \parallel Q]\!] & = & \{(s, X \cup Y)/(s, X) \in [\![P]\!] \wedge (s, Y) \in [\![Q]\!]\} \\
[\![P|||Q]\!] & = & \{(u, X)/\exists s, t.(s, X) \in [\![P]\!] \wedge (t, X) \in [\![Q]\!] \wedge u \in merge(s, t)\} \\
 & & merge(s, t) \text{ es el conjunto de mezclas de las trazas } s \text{ y } t \\
[\![P \backslash a]\!] & = & \{(s \backslash a, X)/(s, X \cup \{a\}) \in [\![P]\!]\} \cup \{((s \backslash a)t, X)/(s, X \cup \{a\}) \\
 & & \in [\![P]\!] \ \wedge \ (t, X) \in [\![CHAOS]\!] \ \wedge \ \forall n \ (sa^n, \emptyset) \in [\![P]\!]\} \\
 & & s \backslash a \text{ resulta de eliminar en } s \text{ las ocurrencias de acciones } a
\end{array}
$$

Cuadro 2.1: Semántica denotacional de CSP

nos indica que el proceso P puede pasar a comportarse, por decisión interna, como el proceso Q, sin que sea visible la ejecución de acción alguna.

La Tabla 2.2 recoge la definición estructurada del conjunto de transiciones operacionales de los procesos, tanto visibles como internas.

Sistema de prueba para CSP

Es posible describir de una forma más compacta la semántica de CSP, por medio de un Sistema de Prueba. El mismo consta de un conjunto de axiomas y reglas, que reflejan las propiedades de cada operador del lenguaje, junto con las interrelaciones entre los mismos. Dichos axiomas definen la noción de *equivalencia semántica* entre procesos, junto con el orden entre los mismos inducido por la relación de orden en que se basa la definición de la semántica denotacional. El uso de esta relación de orden es preceptivo, aún en el caso de que en principio sólo estemos interesados en la equivalencia entre procesos, de cara a incluir en el marco axiomático el tratamiento de los procesos definidos recursivamente.

1)	$\dfrac{}{a \to P \xrightarrow{a} P}$	**2**)	$\dfrac{}{P \sqcap Q \longrightarrow P}$	**3**)	$\dfrac{}{P \sqcap Q \longrightarrow Q}$
4)	$\dfrac{P \xrightarrow{a} Q}{P \sqcap R \xrightarrow{a} Q}$	**5**)	$\dfrac{P \xrightarrow{a} Q}{R \sqcap P \xrightarrow{a} Q}$		
6)	$\dfrac{P \xrightarrow{a} P'}{P \,\Box\, Q \xrightarrow{a} P'}$	**7**)	$\dfrac{Q \xrightarrow{a} Q'}{P \,\Box\, Q \xrightarrow{a} Q'}$	**8**)	$\dfrac{P \longrightarrow P',\ Q \longrightarrow Q'}{P \,\Box\, Q \longrightarrow P' \,\Box\, Q'}$
9)	$\dfrac{P \longrightarrow P'}{P \parallel Q \longrightarrow P' \parallel Q}$	**10**)	$\dfrac{Q \longrightarrow Q'}{P \parallel Q \longrightarrow P \parallel Q'}$	**11**)	$\dfrac{P \xrightarrow{a} P',\ Q \xrightarrow{a} Q'}{P \parallel Q \xrightarrow{a} P' \parallel Q'}$
12)	$\dfrac{P \longrightarrow P'}{P \vert\vert\vert Q \longrightarrow P' \vert\vert\vert Q}$	**13**)	$\dfrac{Q \longrightarrow Q'}{P \vert\vert\vert Q \longrightarrow P \vert\vert\vert Q'}$		
14)	$\dfrac{P \xrightarrow{a} P'}{P \vert\vert\vert Q \xrightarrow{a} P' \vert\vert\vert Q}$	**15**)	$\dfrac{Q \xrightarrow{a} Q'}{P \vert\vert\vert Q \xrightarrow{a} P \vert\vert\vert Q'}$		
16)	$\dfrac{P \longrightarrow Q}{P \setminus a \longrightarrow Q \setminus a}$	**17**)	$\dfrac{P \xrightarrow{a} Q}{P \setminus a \longrightarrow_Q \setminus a}$	**18**)	$\dfrac{P \xrightarrow{b} Q,\ b \neq a}{P \setminus a \xrightarrow{b}_Q \setminus a}$
19)	$\dfrac{P[P/X] \longrightarrow P'}{\mu X.P \longrightarrow P'}$	**20**)	$\dfrac{P[P/X] \xrightarrow{a} P'}{\mu X.P \xrightarrow{a} P'}$		

Cuadro 2.2: Semántica operacional de CSP

El núcleo del sistema lo constituye el conjunto de axiomas correspondiente a los procesos finitos definibles a partir de la constante STOP, por medio del operador prefijo y los dos operadores de elección, que presentamos en la Tabla 2.3, donde también se muestran las reglas generales que nos indican la relación entre la equivalencia y la relación de orden entre procesos, y el hecho de que todos los susodichos operadores son monótonos respecto de dicho orden.

Posteriormente, en la Tabla 2.4 incluimos los axiomas que nos permiten razonar sobre procesos infinitos. Al respecto tenemos por una parte los axiomas correspondientes al proceso *CHAOS* que denota sintácticamente al elemento mínimo del dominio de procesos, y los axiomas correspondientes al operador de recursión.

Por último, en la Tabla 2.5 se reflejan los axiomas correspondientes a los operadores de composición paralela, entrelazamiento y ocultación. Dichos axiomas nos muestran que en todos los casos se trata de operaciones derivadas, por poderse reescribir toda aplicación de las mismas sobre términos construíbles con las operaciones básicas antes consideradas, a un nuevo término en el que sólo aparecen dichas operaciones.

$A1)$	$P \sqcap P \equiv P$	$A7)$	$P \sqcap (Q \square R) \equiv (P \sqcap Q) \square (P \sqcap R)$
$A2)$	$P \square P \equiv P$	$A8)$	$P \square (Q \sqcap R) \equiv (P \square Q) \sqcap (P \square R)$
$A3)$	$P \sqcap Q \equiv Q \sqcap P$	$A9)$	$P \square STOP \equiv P$
$A4)$	$P \square Q \equiv Q \square P$	$A10)$	$P \sqcap Q \sqsubseteq P$
$A5)$	$P \sqcap (Q \sqcap R) \equiv (P \sqcap Q) \sqcap R$	$A11)$	$(a \rightarrow P) \sqcap (a \rightarrow Q) \equiv a \rightarrow P \sqcap Q$
$A6)$	$P \square (Q \square R) \equiv (P \square Q) \square R$	$A12)$	$(a \rightarrow P) \square (a \rightarrow Q) \equiv a \rightarrow P \sqcap Q$
$O1)$	$\dfrac{P \sqsubseteq Q \sqsubseteq P}{P \equiv Q}$	$M1)$	$\dfrac{P \sqsubseteq Q}{a \rightarrow P \ \sqsubseteq \ a \rightarrow Q}$
$O2)$	$\dfrac{P \equiv Q}{P \sqsubseteq Q \sqsubseteq P}$	$M2)$	$\dfrac{P \sqsubseteq Q \ \wedge \ P' \sqsubseteq Q'}{(P \sqcap P') \ \sqsubseteq \ (Q \sqcap Q')}$
$O3)$	$\dfrac{P \sqsubseteq Q \sqsubseteq R}{P \sqsubseteq R}$	$M3)$	$\dfrac{P \sqsubseteq Q \ \wedge \ P' \sqsubseteq Q'}{(P \square P') \ \sqsubseteq \ (Q \square Q')}$

Cuadro 2.3: Axiomas y reglas para procesos finitos

$B1)$	$P \sqcap CHAOS \equiv CHAOS$	
$B2)$	$P \square CHAOS \equiv CHAOS$	
$B3)$	$CHAOS \sqsubseteq P$	
$B4)$	$P[(\mu\xi.P)\|\xi] \sqsubseteq \mu\xi.P$	
$R)$	$\dfrac{\forall Q \in FIN(P) : Q \sqsubseteq R}{P \ \sqsubseteq \ R}$	

Cuadro 2.4: Axiomas y reglas para procesos recursivos

2.4. Extensiones de los lenguajes formales de especificación

En esta sección repasaremos brevemente algunas de las extensiones más habituales que se han realizado para aumentar la expresividad de los lenguajes de especificación formales. Por medio de cada una de dichas extensiones se aumenta igualmente la capacidad de análisis del sistema modelado, especialmente en lo concerniente a la influencia del factor considerado en la extensión correspondiente.

$$
\begin{array}{ll}
PAR0) & P\|CHAOS \equiv CHAOS \\
PAR1) & P\|(Q \sqcap R) \equiv (P\|Q) \sqcap (P\|R) \\
PAR2) & P\|Q \equiv a \rightarrow (P_a\|Q_a) \\
PAR3) & P\|Q \equiv Q\|P \\
\\
INT0) & P|||CHAOS \equiv CHAOS \\
INT1) & P|||STOP \equiv P \\
INT2) & P|||(Q \sqcap R) \equiv (P|||Q) \sqcap (P|||R) \\
INT3) & P|||Q \equiv (\!(a_i \rightarrow (P_i|||Q))\!) \Box (\!(b_j \rightarrow (P|||Q_j))\!) \\
\\
HID0) & CHAOS\backslash b \equiv CHAOS \\
HID1) & (P \sqcap Q)\backslash b \equiv (P\backslash b) \sqcap (Q\backslash b) \\
HID2) & (b \rightarrow P)\backslash c \equiv \begin{cases} (b \rightarrow P\backslash c) & \text{si } b \neq c \\ P\backslash c & \text{si } b = c \end{cases} \\
HID3) & (\hookrightarrow P_a)\backslash c \equiv \begin{cases} a \rightarrow (P_a\backslash c) & \text{si } c \notin A \\ (P_c\backslash c) \sqcap a \rightarrow (P_a\backslash c) & \text{si } c \in A \end{cases}
\end{array}
$$

Cuadro 2.5: Axiomas para procesos derivados

2.4.1. Extensiones probabilistas

Una de las extensiones posibles es la extensión *probabilista*, donde se trata de convertir parte de la incertidumbre debida al indeterminismo en incertidumbre *cuantificable* (véase por ejemplo [GSST90, LS91, BBS95, GSS95, JYL01, CCV+03, Núñ03]). Así, cuando una máquina pueda decidir en un estado entre a y b no tendrá que decidir sin más entre una y otra, sino que, por ejemplo, el 25 por ciento de las veces elegirá a y el 75 por ciento elegirá b. Para representar dicho tipo de situaciones, cada transición ofrecida por la semántica operacional deberá incluir una cierta información asociada (una etiqueta) que denote la probabilidad de que se produzca el evento que representa la transición. Siguiendo la filosofía utilizada en los formalismos presentados anteriormente, dicha información debería en general especificarse en el nivel más básico, es decir, a nivel de la parte (proceso), de tal forma que las consecuencias de la misma a nivel del todo (sistema) puedan deducirse a partir de ella.

La primera restricción que debería cumplirse en principio en la semántica operacional de un lenguaje de especificación que implemente una extensión probabilista es que las probabilidades asociadas a cada una de las transiciones que pueda realizar un sistema desde un estado dado cualquiera sumen 1. No obstante, aplicar tal restricción puede hacer que en algunos casos represente-

mos con una única distribución de probabilidad varios tipos de incertidumbre incomparables. Es más, podríamos correr el riesgo de considerar como cuantificables ciertas incertidumbres que no lo son. Como dijimos en la sección 2.2 en referencia al indeterminismo, podemos distinguir entre indeterminismo a nivel de proceso y a nivel de sistema. Además, dentro del primero podemos considerar una división más fina entre indeterminismo debido al entorno y el debido a acciones internas. Podemos generalizar dicha clasificación para enmarcarla dentro del concepto más general de *incertidumbre*, sea esta indeterminista o cuantificable. Debemos tener en cuenta que cuando un proceso lleva a cabo una acción, dicha acción surge como resultado de la resolución de sus incertidumbres, sean estas debidas a factores externos o internos.

Sin embargo, asumir que podemos asociar una probabilidad con todas las acciones posibles del proceso de tal forma que dichas probabilidades sumen 1 supone asimismo asumir que todas las incertidumbres del proceso son cuantificables. Es más, dado que la probabilidad de dichos sucesos está delimitada unívocamente por el proceso, estaríamos asumiendo igualmente que el motivo de que cada probabilidad sea una dada es inherente a dicho proceso. Sin embargo, en general un proceso no podrá cuantificar la probabilidad de que su entorno escoja una opción de entre varias, pues, de hecho, dicha probabilidad es independiente del proceso en cuestión.

Es por ello que suele distinguirse entre modelos *reactivos* y modelos *generativos* (véase [GSST90, GSS95] donde estas dos interpretaciones de la información probabilista se presentan en el contexto de las álgebras de procesos). En los lenguajes de especificación referidos a modelos reactivos se asume que el entorno puede producir en cada momento una única entrada. Dado que las probabilidades asociadas a las acciones de los procesos sólo pueden deberse a aquellos aspectos sobre los que el proceso tiene influencia, se asocia una distribución de probabilidad para cada estado del proceso *y cada entrada posible*. Es decir, las probabilidades asociadas con cada una de las transiciones que puede producir un proceso desde un mismo estado y ante una misma entrada producida por el entorno deben sumar 1, pero no existe ninguna relación cuantificable probabilísticamente entre las transiciones que se producen como respuesta a entradas distintas, pues dicha relación será controlada únicamente por el entorno.

Por contra, en los modelos *generativos* se asume que el entorno puede ofrecer *más de una entrada* al proceso en cada momento. En dicha situación, el proceso deberá escoger primero cuál de las entradas atiende de entre el conjunto de las que se ofrecen. Una vez fijada la entrada, el proceso escogerá la transición correspondiente a dicha entrada que ejecutará. No obstante,

una vez más las probabilidades asociadas con las transiciones del proceso no podrán relacionar entre sí las entradas que se reciben desde el entorno en un momento determinado con las que *no* se reciben en dicho momento.

Es importante destacar las consecuencias que tienen las consideraciones anteriores a efectos de la incertidumbre que se da a nivel de *sistema*. En caso de que la salida de un proceso sea la entrada externa a otro proceso, entonces la probabilidad de que se dé un determinado evento de comunicación entre dos procesos vendrá delimitada por las respectivas incertidumbres cuantificables a nivel de proceso que se ven involucradas. Por lo tanto, la probabilidad de que se produzca una determinada comunicación entre dos procesos vendrá delimitada por la probabilidad de que el proceso iniciante de la comunicación produzca la salida correspondiente multiplicada por la probabilidad de que el proceso receptor produzca la transición correspondiente como reacción a dicha salida concreta.

También resultará relevante la manera en que manejemos las *acciones internas* a la hora de producir el comportamiento observable de los procesos y los sistemas. Tendremos en cuenta, en general, el hecho de que podremos trasladar las probabilidades asociadas con las acciones internas no observables para que sean contabilizadas dentro de las probabilidades asociadas con las acciones visibles que se producen con posterioridad. Para ello, las probabilidades de cada decisión interna anterior a la acción visible podrán multiplicarse para dar lugar a un nuevo modelo donde las probabilidades de cada acción visible tengan en cuenta las decisiones internas anteriores, de forma que las acciones internas puedan eliminarse del modelo.

2.4.2. Extensiones temporales

Veamos ahora otra de las extensiones más tradicionales de los lenguajes de especificación de sistemas concurrentes. Se trata de la representación del *tiempo* consumido por las acciones ejecutadas en los sistemas [RR88, NS91, QdFA93, AD94, HR98, LdF99, RR99, BM01]. Si bien la semántica operacional de cualquier lenguaje de especificación asume implícitamente una relación de temporalidad entre las acciones, dicha relación se limita habitualmente a representar una relación de simple *posterioridad*, es decir, a representar que una acción se produce después de otra, sin proveer ningún detalle adicional que permita inferir el tiempo concreto consumido en su ejecución. Por contra, el objetivo de las extensiones temporales es representar *explícitamente* el paso del tiempo, de forma que pueda cuantificarse la cantidad de tiempo consumido en cada transición.

De la misma forma que en el caso de las extensiones probabilistas, la información temporal extra puede representarse por medio de etiquetas asociadas a las transiciones que especifiquen el tiempo consumido por la ejecución de cada transición (entendido, por ejemplo, como el tiempo que ha transcurrido desde que se produjo la transición inmediatamente anterior). Así, la reacción de un proceso a una entrada recibida desde el entorno se producirá de manera retardada, representando el hecho de que el proceso ha tenido que llevar a cabo una serie de cómputos que han requerido el consumo de una cierta cantidad de tiempo. El consumo de tiempo en un proceso puede representarse bien por medio de acciones de comunicación que lleven asociado un determinado retardo, bien por medio del uso de dos tipos de acciones distintas.

En este segundo caso distinguimos entre aquellas acciones que representan el envío de un determinado mensaje, que se consideran inmediatas (es decir, no consumen tiempo), y aquellas que representan el paso de tiempo en sí mismo. Así, el tiempo consumido por una determinada acción puede representarse por medio de una acción de retardo seguida de la acción de comunicación en la que se manifiesta externamente la ejecución de dicha acción.

La extracción del comportamiento temporal de los sistemas a partir del comportamiento de los procesos debe tener en cuenta diversos factores. En primer lugar, es necesario considerar que el paso del tiempo en un proceso puede tener distintas consecuencias sobre los demás procesos. En el caso en que estemos modelando un sistema para el que la evolución de los procesos es intercalada y no simultánea (como podría suceder por ejemplo en un sistema monoprocesador) ocurre que el transcurso de tiempo en un proceso no debería tener, en principio, ninguna consecuencia temporal en los demás procesos.

Por contra, si el modelo representa un sistema en el que la evolución de los distintos procesos se produce de manera simultánea (como podría suceder por ejemplo en un sistema multiprocesador) entonces el consumo de tiempo en un proceso deberá conllevar la evolución del tiempo en todos los demás procesos que evolucionan simultáneamente con él. En este último caso, la semántica operacional del lenguaje deberá diseñarse de forma que la *sincronización* entre los procesos no se produzca únicamente a través de la comunicación de mensajes, sino también por el paso del tiempo. Obviamente, en un sistema real la sincronización por el paso del tiempo se produce de forma implícita, sin que se produzca ninguna comunicación adicional. Dicha sincronización representa el simple hecho de que el tiempo transcurre a la vez para todos. Así, el transcurso de 3 segundos en un proceso conllevará el

transcurso de otros tantos segundos en todos los procesos que formen parte del sistema. Si el transcurso del tiempo no se desarrollara de manera sincronizada entre todos los procesos, entonces el orden en que la semántica operacional produce las transiciones correspondientes a las distintas acciones podría no corresponderse con el orden temporal de los eventos tal como serían detectados por un observador externo.

Es más, para lograr que el orden de transiciones se corresponda con la secuencia temporal de sucesos, será necesario que la máxima transición de paso de tiempo del sistema se corresponda con el tiempo mínimo en el que alguno de los procesos del sistema podría ejecutar una acción observable. Ello es así dado que en caso contrario dicha acción podría mostrarse después del verdadero momento en que realmente se produjo.

Una excepción a dicha regla se presenta cuando dos procesos que se ejecutan simultáneamente se comunican entre sí. En tal caso, el momento exacto de la sincronización lo delimitará el proceso que *más tarde* en llegar al punto en que se produce la comunicación, por lo que el otro proceso deberá esperar a dicho momento para manifestar tal comunicación. Como resultado, el instante concreto en el que el proceso más rápido podría ejecutar la acción será irrelevante.

Con respecto a las acciones internas, debemos tener en cuenta que un observador externo no podrá distinguir los retardos concretos producidos por la ejecución de cada uno de ellas. Por ello, desde el punto de vista de dicho observador externo, los distintos retardos consecutivos debidos a acciones internas deberán agruparse para dar lugar a retardos que se producirán únicamente entre cada par de acciones observables consecutivas.

2.4.3. Extensiones estocásticas

El nivel de detalle de la representación del tiempo puede aumentarse para especificar sistemas en los que el tiempo consumido por cada acción no tiene porqué ser siempre el mismo. Es más, podemos representar sistemas en los que es conocida la probabilidad de que el tiempo consumido por una determinada acción en un determinado estado sea uno dado. En caso de que el número de retardos posibles para cada acción y estado sea finito, podremos lograr tales comportamientos por medio de lenguajes híbridos que contemplen probabilidades y tiempo. Así, desde un mismo estado podrían existir dos transiciones salientes correspondientes a la misma acción que, no obstante, consumen tiempos distintos y tienen cada una su propia probabilidad.

Sin embargo, podríamos también desear modelar sistemas en los que el conjunto de tiempos diferentes que puede consumir un sistema para ejecutar una acción dada desde un estado determinado sea infinito. En tal caso, dicho tipo de sistemas híbridos resultaría inútil, pues la descripción del modelo sería igualmente infinita. Es necesario entonces un tipo de sistema que permita expresar de una manera compacta los infinitos tiempos diferentes que podría consumir la ejecución de cada acción, junto con una información probabilista que nos indique la propensión con que se dará cada tiempo posible. Surge así la aparición del *tiempo estocástico* en los lenguajes de especificación de sistemas concurrentes [GHR93, ABC$^+$94, Hil96, BG98, HS00, LN01, HHK02, BG02, LNR02].

Los lenguajes de especificación para la representación de procesos estocásticos permiten asociar con cada acción una variable aleatoria que muestra, para cada valor posible, la probabilidad de que el tiempo consumido por la ejecución de la acción sea menor o igual que dicho valor. Nótese que en el caso de que la función de distribución sea continua no podremos hablar de la probabilidad de que el tiempo transcurrido sea uno determinado pues siempre sería 0. Por ello, una definición en términos de tiempos iguales o menores que cada valor dado es la adecuada.

Al igual que en el caso de las extensiones basadas en la inclusión de tiempos deterministas, la principal dificultad de los lenguajes que permiten la inclusión de información temporal estocástica radica en captar la manera en que el tiempo consumido por un proceso influye en el transcurso de tiempo en los demás procesos.

Nótese que en el caso de los sistemas probabilistas que comentamos en el apartado 2.4.1 la aplicación de cada regla de la semántica operacional, etiquetada por medio de su probabilidad asociada, nos lleva al nuevo estado de tal forma que queda *completamente determinado* el camino que se ha escogido para ir desde el estado anterior hasta el estado actual. Es decir, avanzar por medio de una determinada regla de la semántica operacional supone eliminar la incertidumbre con respecto a la decisión inmediatamente anterior, por lo que en un sistema probabilista el estado actual está completamente determinado y no depende de incertidumbres anteriores que estén sin resolver.

Sin embargo, la situación será habitualmente diferente en un sistema con tiempo definido de forma estocástica. Esto es así porque si la semántica operacional se diseña de forma que cada transición vaya etiquetada con la *variable aleatoria* asociada a dicha transición, entonces el modelo evolucionará por medio de dicha transición a un nuevo estado en el que el tiempo consumido

en la última transición es desconocido *porque no se ha concretado*. Es decir, el estado resultante es lo suficientemente genérico como para representar cualesquiera de los tiempos que se hubieran podido concretar a través de la transición anterior.

Otra perspectiva diferente sería diseñar la semántica operacional del lenguaje de forma que se concretara un tiempo *específico* de entre todos los posibles por medio de la ejecución de la transición. Sin embargo, en tal caso el conjunto de transiciones posibles desde un estado sería infinito, igual que lo es en general el conjunto de tiempos que podría consumir una acción cuando dicho tiempo viene determinado por una variable aleatoria continua. Abrir infinitas ramas desde un estado dado añade una gran complejidad a la hora de verificar la existencia de ciertas relaciones semánticas entre los sistemas, pues, como hemos visto, dichas técnicas se basan en recorrer todos los caminos posibles, buscando concordancias de comportamiento.

Desgraciadamente, en el caso de que las transiciones no concreten los tiempos consumidos y vayan etiquetadas por la variable aleatoria asociada, surgen otros problemas diferentes. Estos aparecen al considerar la composición de procesos para formar sistemas.

Al ir etiquetadas las transiciones por medio de variables aleatorias, en lugar de por tiempos concretos, el *envejecimiento* de cada proceso debido al transcurso de tiempo en otro proceso tampoco podrá expresarse en términos de un tiempo concreto, sino que tendría que hacerse igualmente en términos de una *variable aleatoria* que denote dicho envejecimiento. Pero si permitimos expresar los tiempos estocásticos por medio de *cualquier* tipo de variable aleatoria, entonces la propagación de los efectos de envejecimiento resultará muy compleja, pues no se dispone de un mecanismo que permita describir de una manera *analítica* la variable aleatoria resultante después de aplicarle el envejecimiento producido por otra variable aleatoria cualquiera.

En este contexto, existe un interesante resultado matemático que permite salvar dichas dificultades, si bien a costa de reducir la expresividad a la hora de representar retardos estocásticos. Sucede que una variable aleatoria exponencial utilizada para denotar la longitud de un tiempo determinado no tiene *memoria*, esto es, su distribución no varía con el paso del tiempo. Es decir, si esperamos una cantidad de tiempo y el suceso descrito por la variable aleatoria no se ha producido todavía, entonces la distribución probabilista del tiempo que deberemos seguir esperando desde dicho momento será la misma que en el momento en el que iniciamos la espera.

Esta propiedad permitirá que en los lenguajes de especificación que denoten tiempos estocásticos el envejecimiento de los procesos como resultado de la evolución de otro proceso resulte trivial *siempre que* todas las variables aleatorias se restrinjan a ser exponenciales. Ello es así porque las variables aleatorias de los procesos a envejecer se describirán por medio de la misma variable aleatoria después de ser envejecidas, *independientemente* del tiempo concreto que hayan envejecido. En consecuencia, podremos etiquetar las transiciones del sistema por medio de la variable aleatoria asociada y no con un tiempo concreto, y a pesar de ello las variables aleatorias de los demás procesos tras el transcurso de dicho tiempo *genérico* podrán calcularse de manera sencilla (de hecho, trivial, pues serán sin más las mismas).

Capítulo 3

Álgebra de Procesos Temporizada BTC

3.1. Justificación

Como se ha comentado en el capítulo anterior, la aparición de la concurrencia representa un aumento radical en la complejidad de los sistemas, lo cual se pone de manifiesto en los modelos teóricos desarrollados sucesivamente para abordar su estudio a un nivel formal. Debido a este aumento en la complejidad, los primeros modelos estudiados abstraen todos los aspectos de los procesos concurrentes que se consideran no fundamentales de los mismos, de manera que se pueda razonar a nivel formal sobre el comportamiento de los procesos, sin que la complejidad del modelo formal en cuestión oscurezca y dificulte dichos razonamientos. Naturalmente, el coste que se ha de pagar debido a dichas simplificaciones, es la pérdida de toda aquella información que voluntariamente se ha abstraído en el proceso de modelización.

Ahora bien, una vez estudiados en profundidad los modelos básicos, es oportuno el tratar de ampliarlos o modificarlos en lo preciso, de modo que pueda pasar a estudiarse aquellos aspectos, fundamentalmente cuantitativos, que se habían perdido como resultado de las abstracciones en que aquellos se basaban. Ejemplos de estas extensiones se han presentado en el capítulo anterior, y en concreto extensiones temporizadas como es la nuestra, pero todas ellas presentan la limitación de que asumen máximo paralelismo, esto es, suponen que disponen de tantos procesadores como se requieran, cuando es bien sabido que en los sistemas reales esto no es así. También hemos detectado la falta de atención que se ha prestado a la disponibilidad de los

recursos existentes en el sistema para ser utilizados por los procesos, a pesar de que no cabe duda que una escasez o mala planificación de los recursos tiene consecuencias importantes tanto en la evolución de un sistema como en los resultados temporales que se pueden obtener. Veamos esta problemática más en detalle, cómo se ha abordado desde distintos frentes y por qué pensamos que nuestro lenguaje, que entre otras cosas tiene estos dos aspectos en cuenta, es innovador y a la vez necesario.

Es práctica habitual que al hacer una abstracción de un sistema, éste quede representado como un número de procesos concurrentes que normalmente se están ejecutando en un procesador compartido donde la ejecución de los distintos procesos se va intercalando. Para tales sistemas, el hecho de que los procesos tengan que compartir el procesador no puede ser ignorado ya que influye tanto en el comportamiento temporal como en el funcional del sistema. Sistemas de procesos que comparten un solo procesador se han descrito correctamente en la teoría de scheduling, véase por ejemplo [Liu00]. Sin embargo, las teorías de scheduling generalmente abstraen el comportamiento real de los procesos y por eso no pueden ser usadas para analizar las propiedades de funcionalidad del sistema.

Por otro lado, los modelos operacionales tales como los autómatas y las álgebras de procesos pueden describir el comportamiento funcional del sistema en detalle. Ya ha quedado demostrado que hoy en día, extensiones de estos modelos con noción de tiempo están bien establecidos; sin embargo, las álgebras de procesos temporizadas usualmente asumen máximo paralelismo, lo cual puede ser interpretado como si cada proceso dispusiera de un procesador dedicado. Este hecho hace que no sean adecuados para modelar sistemas de tiempo real donde los procesos comparten un procesador ya que, bajo el supuesto de máximo paralelismo, se entiende que el procesador siempre está disponible para un proceso, lo que significaría que el paso del tiempo puede describirse sin considerar si el procesador está disponible o no cuando evidentemente no es cierto.

El álgebra de procesos que nosotros presentamos (**BTC**) tiene en cuenta el número de procesadores que existen en el sistema y la disponibilidad en cada momento. Con ello se consiguen aunar las características de los modelos algebraicos, que permiten modelar el comportamiento real de los procesos, pero sin suponer máximo paralelismo, con lo que se incluyen los razonamientos hechos en la teoría de scheduling.

La consecuencia más importante de este enfoque es que si hay más procesos que procesadores entonces, no se pueden ejecutar simultáneamente todos los procesos. Un proceso deberá esperar a que le sea asignado un procesador

para poder continuar su ejecución. Con esto llegamos a la conclusión de que en un sistema, y por lo tanto en nuestra especificación de dicho sistema, encontramos dos tipos de esperas en la ejecución de los procesos; las esperas relativas a la sincronización entre procesos y las relativas a la asignación de recursos (entre ellos el procesador). Las primeras son las que habitualmente encontramos en la teoría de concurrencia, mientras que las segundas sólo aparecen al considerar que el número de procesadores de un sistema es limitado y son las esperas que se consideran en la teoría de scheduling.

Con el fin de poder presentar nuestro modelo, creemos que vale la pena revisar una característica importante de los modelos formales de concurrencia. Dichos modelos pueden ser clasificados, en líneas generales, en dos grupos: modelos basados en **interleaving**, en los cuales la ejecución independiente de dos procesos se modela especificando el posible interleaving de sus acciones; y los modelos de **concurrencia real** (true concurrency) en los que las relaciones entre acciones se representan explícitamente. Esto es, la noción de concurrencia real distingue entre comportamiento concurrente y comportamiento en interleaving de la siguiente manera: $a \parallel b \neq a.b + b.a$.

Existen bastantes versiones de álgebras de procesos con tiempos basadas en el modelo de interleaving [BL91, MT90, Sch95] donde las relaciones de equivalencia son normalmente usadas para establecer la corrección de la implementación con respecto a la especificación de los sistemas concurrentes. Y por supuesto también han aparecido algunas álgebras de procesos para expresar concurrencia real [BCHK93, DP95, MY92] considerando localidad y relación entre acciones.

El modelo que nosotros presentamos en esta tesis considera todos los posibles casos entre los modelos basados en interleaving y los basados en concurrencia real. Esto es así por que nosotros podemos trasladar a nuestra especificación cuál es el número de procesadores de los que dispone un sistema, por lo tanto si consideramos que se dispone de tan sólo un procesador, estaremos trabajando intercalando procesos, esto es en interleaving. Si incluimos en la especificación el número exacto de procesadores existentes (y es mayor que uno) estaremos trabajando con concurrencia real, mientras que si consideramos que en el sistema hay una existencia ilimitada de procesadores (infinitos) estaremos trabajando del mismo modo que los modelos que trabajan con máximo paralelismo.

Pero nuestro lenguaje no se limita a tener en cuenta el número de procesadores disponibles en el sistema, sino que generalizamos y somos capaces de considerar cualquier tipo de recurso que un proceso pueda necesitar para su ejecución. Así, el análisis de prestaciones de nuestro sistema será diferente

dependiendo del número de recursos disponibles en cada momento. Si denotamos por $\{\![P]\!\}_N$ a un proceso que se ejecuta en un sistema que cuenta con N recursos (suponiendo, en una primera aproximación, que todos los recursos del sistema fueran homogéneos), tendremos que $\{\![a|b]\!\}_1 \equiv \{\![a.b + b.a]\!\}_1$ pero $\{\![a|b]\!\}_n \not\equiv \{\![a.b+b.a]\!\}_n \; \forall n > 1$. En general, dado un proceso P, $\forall n, m \in \mathbb{N}$ con $n \neq m$, no podemos saber si los resultados obtenidos al hacer la evaluación de prestaciones de $\{\![P]\!\}_n$ serán iguales a los de $\{\![P]\!\}_m$. De hecho, una de las principales aplicaciones de nuestra álgebra de procesos consiste precisamente en encontrar un número natural n tal que $\forall i \in \mathbb{N}, i \geq n$, $\{\![P]\!\}_i$ es *equivalente* (desde un punto de vista de análisis de prestaciones temporales) a $\{\![P]\!\}_n$, esto es, n representa el máximo grado de paralelismo al que podemos someter el sistema, y como resultado lo que realmente obtenemos es el número óptimo de recursos necesarios para obtener los mejores valores en la evaluación de prestaciones del sistema.

BTC está basada en CSP. Nosotros hemos extendido su sintaxis con el fin de poder considerar la duración de las acciones mediante el operador prefijo temporizado. La semántica operacional también ha necesitado ser extendida para poder tener en cuenta el contexto (número y tipo de recursos) en el que los procesos se ejecutan. Además hemos necesitado introducir la noción de bolsa de acciones para ser capaces de representar la ejecución simultanea de varias acciones.

En la literatura encontramos varios modelos teóricos que consideran también la noción de consumo de recursos. Algunas álgebras de procesos temporizadas que tienen en cuenta los recursos presentes en el sistema se pueden encontrar en [Gru96, Gru97, BAL02, LBGG94, BGL97, LCK+01]. En [Gru96, Gru97] se presenta un álgebra de procesos temporal con paralelismo limitado (CCSLP). El tiempo se asocia a las acciones considerando acciones especiales t_n en las que n representa el consumo de n unidades de recurso (homogéneo). Esto significa que, si queremos especificar una acción que consume dos unidades de tiempo, debemos escribir $a.t_1.t_1.NIL$, donde a no representa la ejecución de una acción sino tan solo el momento en el cual la acción comienza su ejecución.

Desde nuestro punto de vista, el principal problema de este modelo radica en la dificultad de saber cuando realmente finaliza una acción. Si consideramos tres acciones (a, b, c) con duración 2 (esto es, $a.t_1.t_1.NIL$, $b.t_1.t_1.NIL$, $c.t_1.t_1.NIL$), y dos recursos (procesadores), una posible secuencia de acciones es $a.b.t_2.c.t_2.t_2$, donde el primer t_2 representa la ejecución de la primera unidad de a y b, el segundo t_2 es la ejecución de la primera unidad de c y la segunda unidad de a o b, y el último t_2 representa la ejecución de la segunda

unidad de c y la segunda unidad de b o a. Con este pequeño ejemplo queda claro la dificultad que se presenta al intentar analizar las prestaciones de un sistema cuando es imposible saber en que momento el sistema alcanza un determinado estado.

En [BAL02] se presenta un álgebra de procesos capaz de manejar el hecho de que los procesadores deben de ser compartido por los procesos que se encuentran en el sistema y este reparto se hace de acuerdo a una política de scheduling. Es posible describir sistemas de multiprocesadores pero sólo con asignación estática de los procesos a los procesadores, esto es, una vez que un proceso ha comenzado su ejecución en un procesador que se le ha asignado no podrá cambiar a otro que estuviera libre en las posteriores ocasiones en que dicho proceso necesitara hacer uso del recurso procesador. Esta situación no se da en nuestro modelo ya que no existe asignación de los procesos a los procesadores. Esta diferencia es fundamental ya que permite que un proceso que interrumpe su utilización del procesador podrá más tarde reanudar su ejecución en cualquier otro procesador (o en el mismo) que se encuentre disponible.

Un álgebra de procesos creada para la especificación y análisis de sistemas de tiempo real con recursos limitados, ACSR, se puede encontrar en [LBGG94]. En este enfoque, la existencia de recursos compartidos (heterogéneos) se representa mediante acciones temporizadas y la sincronización se soporta mediante eventos instantáneos. Las acciones temporales son siempre ejecutadas durante una unidad de tiempo (*tic*) y consumen un conjunto de recursos durante ese tiempo. ACRS también permite prioridades que son usadas para decidir que proceso (acción) se selecciona de entre las que compiten por un recurso.

En [BGL97] podemos ver una extensión de ACSR que considera tiempo continuo. En este modelo, las acciones temporales tienen una duración que viene dada por un número real finito y positivo u, donde A^u representa la ejecución de una acción A que consume recursos durante un tiempo u. La sincronización entre acciones temporizadas se permite sólo en el caso de que ambas acciones tengan la misma duración y utilicen un conjunto disjunto de recursos. Otras extensiones de ACSR pueden encontrarse en [LCK$^+$01] donde se presenta la versión probabilista (PACSR) y la versión que utiliza el paso por valor (ACSR-VP).

El enfoque tomado en ACSR y sus extensiones a pesar de ser parecido al nuestro presenta importantes diferencias. En ACSR, una acción temporizada es un conjunto de pares (*recurso*, *prioridad*), y dos (o más) acciones temporizadas pueden ejecutarse simultáneamente sólo en el caso de que usen

recursos distintos. Consideremos un sistema con dos procesadores. Si tenemos en el sistema tres procesos que usan diferentes recursos, entonces los tres procesos podrían ser ejecutados simultáneamente, pero, en realidad sólo contamos con dos procesadores. Para poder considerar que existen tan solo dos procesadores disponibles, necesitamos tomarlos como dos recursos `cpu1`y `cpu2` (en ACSR no se permite tener dos unidades de un mismo recurso). Como ejemplo consideremos los siguientes tres procesos (en sintaxis de ACSR): $P = \{(cpu1, 1)\}^1 : NIL$, $Q = \{(cpu2, 1)\}^1 : NIL$ y $R = \{(cpu1, 1)\}^1 : NIL$. Con ellos podemos ver que en un primer paso P y Q o R y Q pueden evolucionar simultáneamente, pero no P y R porque comparten el recurso `cpu1`. En otras palabras, los recursos vuelven a ser asignados estáticamente a los procesos lo que no ocurre en nuestro modelo donde se realiza una asignación dinámica de recursos en todo momento.

Encontramos un enfoque diferente en [NR01] donde se presenta un álgebra de procesos para el manejo de recursos en sistemas concurrentes (PARM). En este artículo se trabaja en un contexto con recursos heterogéneos y se trabaja con una política de explotación de recursos basada en funciones de utilidad. La idea principal es que varios procesos pueden intercambiar recursos con el fin de mejorar el rendimiento del sistema global siempre y cuando ninguno empeore después del cambio. A pesar de ser un trabajo muy interesante, no está demasiado cerca de nuestro enfoque ya que no se cuantifica en ningún momento el rendimiento del sistema. Un trabajo similar se encuentra en [KCS02] donde se intenta presentar un esquema para una óptima asignación de recursos en redes multiclase.

3.2. Sintaxis del álgebra de procesos BTC

En esta sección introduciremos el álgebra de procesos **BTC** mediante su sintaxis y una interpretación intuitiva de sus operadores.

Definición 1 Sea $\mathcal{A}ct_T$ un conjunto finito de acciones temporizadas y $\mathcal{A}ct_U$ un conjunto finito de acciones sin tiempo, $\mathcal{A}ct_U \cap \mathcal{A}ct_T = \emptyset$. La sintaxis de los procesos del lenguaje viene definida por la siguiente expresión BNF:

$$P ::= stop \mid a.P \mid \ <b, \alpha>.P \mid P \ \oplus \ P \mid P \ + \ P \mid P \parallel_A P \mid recX.P$$

donde $A \subseteq \mathcal{A}ct_U$, $a \in \mathcal{A}ct_U$, $b \in \mathcal{A}ct_T$, $\alpha \in \mathbb{N}$ y $\mathbb{N}$, como es habitual, representa el conjunto de números naturales. Además supongamos un conjunto de variables de procesos Id que toma su valor en el rango X, X', un conjunto de procesos P con valores en P, Q, R y un conjunto de acciones $\mathcal{A}ct = \mathcal{A}ct_U \cup \mathcal{A}ct_T$.

A continuación se hace una descripción informal e intuitiva de los operadores del lenguaje. Posteriormente dotaremos al lenguaje de una semántica operacional, con lo cual quedará formalizado el significado de cada operador, así como la relación existente entre ellos.

stop .- Simboliza una máquina que no puede realizar ninguna acción.

Prefijo. $(a.P)$.- Es el operador prefijo clásico. Representa un proceso P prefijado por una acción sin tiempo. Como es habitual, el proceso $a.P$, una vez que ha ejecutado la acción a se comporta como el proceso P.

Prefijo temporizado. $(< b, \alpha > .P)$.- Al operador prefijo habitual le hemos añadido información sobre la cantidad de tiempo que tarda la acción b en ejecutarse (α). Así, el proceso $< b, \alpha > .P$ una vez ejecutada la acción b pasa a comportarse como el proceso P.

Elección interna. $(P_1 \oplus P_2)$.- Su interpretación es la habitual. Dados dos procesos P_1 y P_2, $P_1 \oplus P_2$ es el proceso que se comporta como P_1 o como P_2, siendo esta elección una decisión interna del sistema.

Elección externa. $(P_1 + P_2)$.- También adoptaremos la interpretación usual para este operador. Dados dos procesos P_1 y P_2, $P_1 + P_2$ es el proceso que se comporta como P_1 o como P_2 dependiendo de lo que el medio externo solicite.

Paralelo. $(P_1 \parallel_A P_2)$.- Representa la ejecución paralela de los procesos P_1 y P_2 los cuales deben sincronizar para ejecutar las acciones del conjunto A, mientras que pueden ejecutar independientemente las acciones que no pertenezcan a dicho conjunto. Como abreviatura notaremos con $P_1 \parallel P_2$ para indicar la ejecución paralela de los procesos P_1 y P_2 en la cual no existe sincronización, esto es, $P_1 \parallel_{\{\}} P_2$.

Recursión. $(recX.P)$.- Tiene la interpretación habitual, similar a la que podemos ver en otras álgebras de procesos como CSP y EPL, es decir, es un proceso en el que las ocurrencias de X se sustituyen por el propio proceso recursivo. Lo utilizaremos para la obtención de procesos infinitos.

Durante esta tesis la notación que utilizaremos para trabajar con multiconjuntos será la siguiente: dado un conjunto X, el conjunto de multiconjuntos que pueden formarse con los elementos de X será denotado por $B(X)$,

y dado un conjunto B sobre X indicaremos el número de instancias de cada elemento de X en B de la forma siguiente:

$$B = \{2.x_1,\ 3.x_2,\ 5.x_3\}$$

que representa el multiconjunto definido sobre el conjunto $X = \{x_1,\ x_2,\ x_3,\ x_4\}$ donde se define la siguiente función:

$$B : \{x_1,\ x_2,\ x_3,\ x_4\} \longrightarrow \mathbb{N}$$

que para el multiconjunto B anterior resultaría:

$$B(x_1) = 2,\ B(x_2) = 3,\ B(x_3) = 5,\ B(x_4) = 0$$

Utilizaremos las letras B, B_1, B_2, C, C_1, C_2 para representar los multiconjuntos, mientras que usaremos A, A_1, A_2 para representar conjuntos. Representaremos por $\emptyset$ tanto el conjunto como el multiconjunto vacío, pues quedará claro por el contexto de qué se trata en cada momento.

También utilizaremos los siguientes operadores sobre multiconjuntos:

- $B_1 \cup B_2$: Es la unión de multiconjuntos, esto es:
  $$(B_1 \cup B_2)(x) = B_1(x) + B_2(x),\ \forall x \in X$$
- $|B|$: Es la cardinalidad de un multiconjunto y es el número de acciones contenidas en dicho multiconjunto. Es decir:
  $$|B| = \sum B(x), \qquad \forall x \in X$$
- $B \cap A$: Es el multiconjunto obtenido al tomar de B únicamente las acciones que pertenecen al conjunto A, es decir:
  $$(B \cap A)(x) = \begin{cases} 0 & \text{si } x \notin A \\ B(x) & \text{en caso contrario} \end{cases}$$

Definición 1 (Bolsa) Definimos una bolsa de acciones B como un multiconjunto definido sobre elementos de Act. □

Definición 2 (Bolsa de acciones temporizadas) Sea $\mathcal{B}(X)$ el conjunto de multiconjuntos que pueden formarse con los elementos de X. Una bolsa de acciones temporizadas es un par (B, α), donde $B \in \mathcal{B}(X)$, y $\alpha \in \mathbb{N}$ indica el tiempo que tarda en ejecutarse cada una de las acciones en B. El conjunto de bolsas de acciones temporizadas sobre el conjunto X, se denota por $TB(X)$, y se define como $TB(X) = \{(B, \alpha) \mid B \in \mathcal{B}(X) \wedge \alpha \in \mathbb{N}\}$. □

Intuitivamente, una bolsa de acciones temporizadas relaciona un multiconjunto con un número natural que representa el tiempo que cada acción de ese multiconjunto necesita para finalizar su ejecución. Todas las acciones incluidas en una misma bolsa emplean el mismo tiempo en su ejecución.

3.3. Semántica Operacional

Con la definición de la semántica operacional, proporcionamos a los operadores del lenguaje un significado e interpretación precisa mediante la descripción de como un proceso puede transformarse en otro proceso. Esta evolución se presenta mediante un sistema de transiciones.

Definición 3 (Transición)

Una transición es un tupla $(P, Q, (B, \alpha))$ que representaremos como:

$$P \xrightarrow{B,\alpha} Q$$

Intuitivamente, el significado de la transición anterior es que el proceso P ejecuta las acciones contenidas en la bolsa B y pasa a comportarse como el proceso Q. Todas las acciones de una misma bolsa requieren el mismo tiempo para su ejecución, en este caso α.

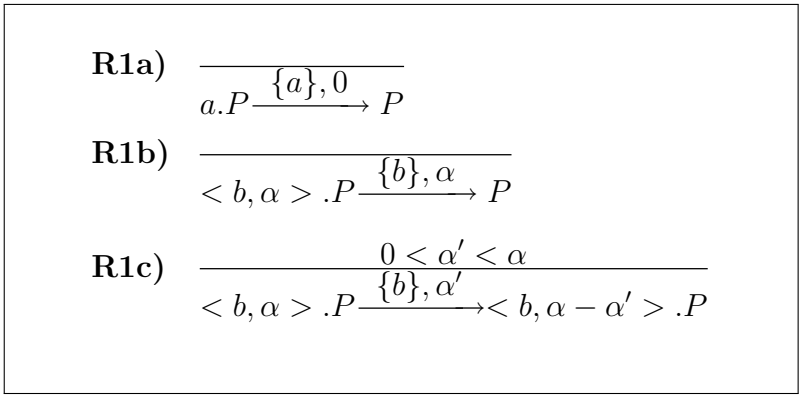
□

Con el fin de poder considerar evoluciones internas (no observables) de un proceso, hemos considerado un nuevo tipo de acción o incluida en la sintaxis: $\tau \notin \mathcal{A}ct$, la cual representa una acción interna. Esta acción se usa para definir la evolución de una elección interna de dos procesos y tiene una duración de 0 unidades de tiempo.

En las Tablas 3.1, 3.2, 3.3, 3.4, y 3.5 podemos ver las reglas para este lenguaje, las cuales pasamos a explicar intuitivamente a continuación. Suponemos que $\alpha, \alpha' \in \mathbb{N}$, $a \in \mathcal{A}ct_U$, $b \in \mathcal{A}ct_T$, $B_i \in \mathcal{B}(\mathcal{A}ct \cup \{\tau\})$ y $B' \in \mathcal{B}(\mathcal{A}ct_U)$. Además también consideramos una constante $\mathcal{N}$ que representa la cantidad de recursos que el sistema tiene disponibles.

Sean los procesos P, Q, P', Q'; las acciones $a, b \in \mathcal{A}ct$; $\alpha_i \in \mathbb{N}$; $B_i \in \mathcal{P}(\mathcal{A}ct)$ y $A_i \subseteq \mathcal{A}ct$.

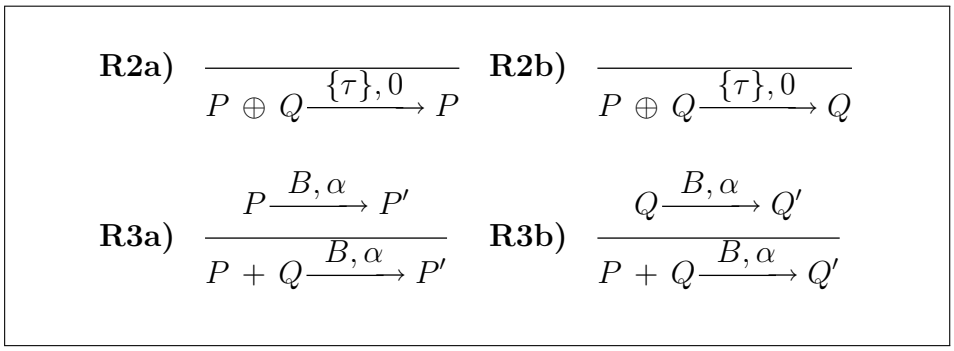
La regla $R1a$ representa el operador prefijo clásico. La regla $R1b$ es el operador prefijo temporizado y establece que dado un proceso P y una acción b que tiene asociado un tiempo de ejecución α, primero se ejecuta b durante α unidades de tiempo después de lo cual el proceso se comportará como P. La regla $R1c$ también representa el prefijo temporizad y tiene una especial importancia ya que con ella permitimos una ejecución parcial de una acción, esto es, si tenemos una acción con una duración α, podemos ejecutarla durante α' unidades de tiempo y dejar el resto de su ejecución (que durará $\alpha - \alpha'$ unidades de tiempo) para ser llevada a cabo más tarde.



Cuadro 3.1: Semántica Operacional: Prefijo y Prefijo Temporizado

Con esta regla $R1c$ estamos permitiendo que una transición con el operador prefijo pueda ser dividida en cualquier número (finito) de transiciones consecutivas. Es una regla parecida a la regla $ActT$ que se presenta en [BGL97] pero con una importante diferencia; en el caso de $ActT$ el dominio considerado para el tiempo es continuo por lo que se pueden derivar un número infinito de transiciones. En nuestro modelo este hecho no puede ocurrir ya que trabajamos con tiempo discreto.

La regla $R2$ y $R3$ son las clásicas para la elección externa e interna por lo que no necesitan más comentario.



Cuadro 3.2: Semántica Operacional: Elección Interna y Elección Externa

La regla $R4$ muestra el mecanismo de sincronización adoptado en nuestro modelo. En ella vemos que sólo se ejecutarán las acciones de sincronización del conjunto A que están a la vez presente en el proceso P y Q. Debido a que las acciones de sincronización son acciones sin tiempo, el proceso completo evoluciona en 0 unidades de tiempo a $P' \parallel_A Q'$.

$$\textbf{R4)}\quad \frac{P \xrightarrow{B',0} P' \ \wedge \ Q \xrightarrow{B',0} Q' \ \wedge \ B' = B' \cap A \neq \emptyset}{P \parallel_A Q \xrightarrow{B',0} P' \parallel_A Q'}$$

Cuadro 3.3: Semántica Operacional: Paralelismo (Sincronización)

La regla $R5a$ captura la ejecución simultánea de como mucho $\mathcal{N}$ acciones que no sean de sincronización, donde recordemos que $\mathcal{N}$ representa el número máximo de recursos con los que cuenta el sistema. Por lo tanto, con esta regla estamos controlando el contexto en el que se ejecutan los procesos (el número de recursos) y no se permite la ejecución simultanea de más acciones que necesiten recursos que los disponibles.

$$\textbf{R5a)}\quad \frac{P \xrightarrow{B_1,\alpha} P' \ \wedge \ Q \xrightarrow{B_2,\alpha} Q' \ \wedge \ B_i \cap A = \emptyset \ \wedge \ |B_1| + |B_2| \leq \mathcal{N}}{P \parallel_A Q \xrightarrow{B_1 \cup B_2,\alpha} P' \parallel_A Q'}$$

$$\textbf{R5b)}\quad \frac{P \xrightarrow{B,\alpha} P' \ \wedge \ B \cap A = \emptyset}{P \parallel_A Q \xrightarrow{B,\alpha} P' \parallel_A Q}$$

$$\textbf{R5c)}\quad \frac{Q \xrightarrow{B,\alpha} Q' \ \wedge \ B \cap A = \emptyset}{P \parallel_A Q \xrightarrow{B,\alpha} P \parallel_A Q'}$$

Cuadro 3.4: Semántica Operacional: Paralelismo

A pesar de que hemos considerado como premisa que la duración de las acciones en las bolsas de los dos procesos participantes tiene que ser igual, esto no supone ninguna pérdida de generalidad ya que aplicando la regla $R1c$ podemos partir la ejecución de las acciones como mejor nos convenga con el fin de obtener el mismo tiempo en las dos premisas dejando para una ejecución posterior el resto de su ejecución.

Las reglas $R5b$ y $R5c$ son similares a la regla $R5a$ pero considerando que sólo uno de los dos procesos evoluciona (P o Q).

Finalmente, la regla $R6$ captura la semántica de la recursión al modo clásico.

$$\textbf{R6)}\quad \frac{P\{recX.P/X\} \xrightarrow{B,\alpha} P'}{recX.P \xrightarrow{B,\alpha} P'}$$

Cuadro 3.5: Semántica Operacional: Recursión

Antes de pasar a la siguiente sección veamos unos elementales ejemplos con el fin de mostrar de una manera sencilla lo recogido en nuestra semántica. En un abuso de notación con el fin de conseguir mayor claridad en ellos, permitiremos que en el operador prefijo temporizado pueda aparecer una bolsa de acciones y no sólo una acción como se recoge en la sintaxis. Con esto logramos representar la idea intuitiva de que las acciones pertenecientes a ese conjunto están preparadas para ejecutarse y que esta ejecución se puede realizar en paralelo (si la disponibilidad de recursos lo permite). Esto no contradice en ningún momento nuestra sintaxis ya que el mismo resultado se consigue con dos procesos en paralelos que contengan esas acciones temporizadas.

Ejemplo 1 Al ejecutar dos procesos en paralelo, en las premisas hemos considerado que las acciones en las bolsas deben de tener igual duración y comentábamos que esto no conlleva una pérdida de generalidad, ya que aplicando la regla $R1c$ podemos partir la ejecución de una acción como mejor nos convenga.

Supongamos que tenemos en paralelo un proceso $P =< \{a,b\}, 2 > .P'$ y otro $Q =< c, 5 > .Q'$. Para el ejemplo no es necesario saber que tipo de recurso es el compartido, pero si necesitamos saber que existen 3 unidades del recurso compartido. Esto quiere decir que al mismo tiempo se podrán estar ejecutando un máximo de 3 acciones que necesiten dicho recurso. Suponemos que todas las acciones de este sistema necesitan de ese recurso (caso bastante realista si, por ejemplo, el recurso fuera un procesador). La evolución del sistema sería la siguiente:

$$\begin{array}{lll} P =< \{a,b\}, 2 > .P' & B_1 = \{a,b\} & |B_1| = 2 \\ Q =< c, 5 > .Q' & B_2 = \{c\} & |B_2| = 1 \end{array}$$

$$\mathcal{N} = 3 \qquad |B_1| + |B_2| = 3 \leq \mathcal{N}$$

$$< \{a,b\}, 2 > .P' \parallel < c, 5 > .Q' \xrightarrow{\{a,b,c\}, 2} P' \parallel < c, 3 > .Q'$$

Vemos que las tres acciones se pueden ejecutar simultáneamente ya que hay suficientes recursos (o instancias del recurso) para atenderlas. Además comprobamos que el hecho de que la duración de las acciones en el paralelo fueran diferentes no presenta problemas ya que la acción más larga se ha ejecutado durante las mismas unidades de tiempo que la más corta y deja para un siguiente paso las unidades restantes.

□

Ejemplo 2 Cuando el número de acciones preparadas para ejecutarse y que necesitan de un recurso compartido es superior de los recursos disponibles en ese momento, aparecen distintas alternativas en la evolución del sistema. Todas estas alternativas deberán ser tenidas en cuenta a la hora de hacer cualquier estudio de prestaciones del sistema.

Supongamos el siguiente procesos ejecutándose en paralelo:

$$\begin{array}{lll} P =< \{a,b\},3 > .P' & B_1 = \{a,b\} & |B_1| = 2 \\ Q =< \{c,d\},2 > .Q' & B_2 = \{c,d\} & |B_2| = 2 \end{array}$$

$$\mathcal{N} = 3 \qquad |B_1| + |B_2| = 4 > \mathcal{N}$$

En este supuesto, tenemos 4 acciones preparadas para ejecutarse pero sólo se podrán ejecutar al mismo tiempo un máximo de 3 con lo que las posibles evoluciones que podría tener este sistema (seleccionando siempre el máximo número de acciones permitidas) serían:

1. $< \{a,b\},3 > .P' \parallel < \{c,d\},2 > .Q' \xrightarrow{\{a,b,c\},2}$
   $< \{a,b\},1 > .P' \parallel < d,2 > .Q' \xrightarrow{(a,b,d),1} P' \parallel < d,1 > .Q'$

2. $< \{a,b\},3 > .P' \parallel < \{c,d\},2 > .Q' \xrightarrow{\{a,b,d\},2}$
   $< \{a,b\},1 > .P' \parallel < c,2 > .Q' \xrightarrow{(a,b,c),1} P' \parallel < c,1 > .Q'$

3. $< \{a,b\},3 > .P' \parallel < \{c,d\},2 > .Q' \xrightarrow{\{a,c,d\},2}$
   $< \{a,b\},1 > . < b,2 > .P' \parallel Q'$

4. $< \{a,b\},3 > .P' \parallel < \{c,d\},2 > .Q' \xrightarrow{\{b,c,d\},2}$
   $< \{b,a\},1 > . < a,2 > .P' \parallel Q'$

□

3.4. Evaluación de prestaciones

Con el modelo formal que hemos definido capturamos las características temporales del sistema a analizar. Ahora nos ocuparemos de definir un algoritmo que nos permita calcular el tiempo necesario para evolucionar entre estados. Para ello, a partir de la semántica operacional podemos construir para cada proceso un grafo de transiciones que nos ayudará en la evaluación formal de prestaciones. En este grafo abstraeremos la información sobre las acciones y sólo consideraremos información sobre el tiempo (duración de acciones). El grafo que obtenemos es un grafo dirigido con pesos, donde los pesos son siempre números positivos.

Una vez obtenido el grafo lo que pretendemos resolver es un problema de maximizar el rendimiento relativo a minimizar el tiempo necesario para alcanzar el estado final (a partir del inicial). Para ello debemos encontrar el camino más corto (con menos peso) entre estos dos estados y lo hacemos utilizando una modificación del algoritmo de Dijkstra.

Denotamos por T_{Si} al tiempo necesario para evolucionar del *estado inicial* S_0 al estado S_i y $In(S_i)$ al conjunto de nodos predecesores del nodo S_i. Para cada estado $S_j \in In(S_i)$ existe un arco etiquetado con α_{ji} que representa el tiempo necesario para que el sistema evolucione del estado S_j al estado S_i. Una representación gráfica se presenta en la Figura 3.1.

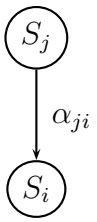


Figura 3.1: Grafo de Transiciones

A partir de aquí, T_{S_i} se puede obtener de forma recursiva usando la siguiente ecuación:

$$T_{S_i} = \begin{cases} 0 & \text{if } S_i = S_0 \\ \min\{T_{S_j} + \alpha_{ji} \mid S_j \in In(S_i)\} & \text{c.c.} \end{cases} \tag{3.1}$$

Con el objetivo de calcular (3.1), numeramos los n nodos del grafo desde 0 a $n-1$ y suponemos que dicho grafo G está representado mediante su matriz

de adyacencias donde $cost[i][j]$ es el peso del arco (i, j) y que los pesos de los arcos que no pertenecen al grafo están inicializados a ∞. Esto es:

$$cost[i][j] = \begin{cases} \alpha_{ij} & \text{if } S_i \in In(S_j) \\ \infty & \text{c.c.} \end{cases}$$

Sea S el conjunto de nodos (incluyendo S_0) para los que ya se ha calculado el mínimo tiempo necesario para llegar a ellos desde el nodo inicial. Con ello, el algoritmo de evaluación de prestaciones quedará como se muestra en la Figura 3.2.

```
TiempoMin(int v, int cost[][SIZE], int n, int dist[]) {
// v es el nodo inicial
// cost es la matriz de adyacencias con pesos
// n es el número de nodos en el grafo
int u, minima; bool S[SIZE];

for (int i=0; i<=n-1; i++) {
    S[i] = false; dist[i] = cost[v][i];
} S[v] = true; dist[v] = 0; for (int num=1; num<=n-1; num++) {
// Elegimos u de entre los nodos que no están todavía en S
// el que tenga dist[u] mínima.
    minima = numero_grande;
    for (int i=1; i<=n-1; i++)
      if ((S[i]==false) && (dist[i]<minima))
            minima=dist[i]; u=i;
    S[u]=true;
    for (int w=1; w<=n; w++) // Actualizamos las distancias
        if ((S[w]==false) && (dist[w]>dist[u]+cost[u][w]))
            dist[w] = dist[u] + cost[u][w];
    }
}
```

Figura 3.2: Algoritmo de Evaluación de Prestaciones

3.5. Un ejemplo: Suma de matrices

Como primer ejemplo vamos a modelar y analizar un algoritmo típico de suma de matrices. Consideraremos que contamos con el mismo número de procesos que filas (columnas) tengan las matrices a sumar. Esta suposición no resta generalidad al ejemplo ya que en caso de ser menor el número de filas que de procesos, simplemente habría procesos que no se ejecutarían y en caso contrario, si el número de filas fuera mayor, los procesos, una vez sumada una fila, comprobarían si quedan filas por sumar en cuyo caso realizaría la operación. Así, nuestro sistema consistirá en un conjunto de procesos ejecutandose en paralelo donde cada uno de ellos estará encargado de realizar la suma de una fila de las matrices (podría hacerse de igual modo por columnas). Esta suma consistirá en tres acciones; lectura de la fila, realización de la suma propiamente dicha y escritura del resultado.

Este algoritmo puede ser modelado con nuestra álgebra como sigue:

$$\text{SumMatriz} \equiv P_1||P_2||...||P_i$$

$$P_i \equiv < rd_i, 4 > . < add_i, 1 > . < wr_i, 2 >$$

donde rd_i es una acción del tipo "leer fila", add_i es del tipo "sumar fila" y wr_i es "escribir fila".

Una vez contamos con la especificación del sistema, el siguiente paso consiste en generar el grafo de transición de estados correspondiente a esta especificación. Como era de esperar el número de nodos del grafo, incluso de un sistema pequeño como el del ejemplo, es bastante elevado por lo que vamos a hacer algunas simplificaciones a la hora de representarlo con el fin de obtener la mayor claridad posible.

Empezaremos con un sistema sencillo que sería el que realiza la suma de dos matrices con 3 filas cada una. El recurso compartido aquí será el procesador y suponemos que el sistema cuenta con 2 procesadores ($\mathcal{N} = 2$). Al realizar el grafo de transiciones se observa que existen tres ramas simétricas a partir del nodo inicial y hemos decidido representar sólo una de ellas. También, por claridad en la representación, se ha seguido la política de coger siempre el mayor número de acciones a realizar simultáneamente en cada momento. Con estas simplificaciones (sólo a la hora de la representación) el grafo de transiciones que se obtiene es el mostrado en la Figura 3.3, donde la correspondencia de cada estado es la siguiente:

$N_0 =< rd_1, 4 > . < add_1, 1 > . < wr_1, 2 > ||$
$\quad < rd_2, 4 > . < add_2, 1 > . < wr_2, 2 > ||$
$\quad < rd_3, 4 > . < add_3, 1 > . < wr_3, 2 >$
$N_1 =< add_1, 1 > . < wr_1, 2 > || < add_2, 1 > . < wr_2, 2 > ||$
$\quad < rd_3, 4 > . < add_3, 1 > . < wr_3, 2 >$
$N_2 =< wr_1, 2 > || < wr_2, 2 > || < rd_3, 4 > . < add_3, 1 > . < wr_3, 2 >$
$N_3 =< add_1, 1 > . < wr_1, 2 > || < wr_2, 2 > ||$
$\quad < rd_3, 3 > . < add_3, 1 > . < wr_3, 2 >$
$N_4 =< wr_1, 2 > || < add_2, 1 > . < wr_2, 2 > ||$
$\quad < rd_3, 3 > . < add_3, 1 > . < wr_3, 2 >$
$N_5 =< rd_3, 4 > . < add_3, 1 > . < wr_3, 2 >$
$N_6 =< wr_2, 2 > || < rd_3, 2 > . < add_3, 1 > . < wr_3, 2 >$
$N_7 =< wr_1, 2 > || < rd_3, 2 > . < add_3, 1 > . < wr_3, 2 >$
$N_8 =< wr_1, 2 > || < wr_2, 1 > || < rd_3, 3 > . < add_3, 1 > . < wr_3, 2 >$
$N_9 =< add_1, 1 > . < wr_1, 2 > || < rd_3, 1 > . < add_3, 1 > . < wr_3, 2 >$
$N_{10} =< wr_1, 2 > || < wr_2, 2 > || < rd_3, 2 > . < add_3, 1 > . < wr_3, 2 >$
$N_{11} =< add_2, 1 > . < wr_2, 2 > || < rd_3, 1 > . < add_3, 1 > . < wr_3, 2 >$
$N_{12} =< wr_1, 1 > || < wr_2, 2 > || < rd_3, 3 > . < add_3, 1 > . < wr_3, 2 >$
$N_{13} =< wr_2, 1 > || < rd_3, 1 > . < add_3, 1 > . < wr_3, 2 >$
$N_{14} =< wr_1, 1 > || < rd_3, 3 > . < add_3, 1 > . < wr_3, 2 >$
$N_{15} =< wr_1, 2 > || < add_3, 1 > . < wr_3, 2 >$
$N_{16} =< wr_2, 2 > || < add_3, 1 > . < wr_3, 2 >$
$N_{17} =< wr_2, 1 > || < rd_3, 3 > . < add_3, 1 > . < wr_3, 2 >$
$N_{18} =< wr_1, 1 > || < rd_3, 1 > . < add_3, 1 > . < wr_3, 2 >$
$N_{19} =< wr_2, 2 > || < rd_3, 2 > . < add_3, 1 > . < wr_3, 2 >$
$N_{20} =< rd_3, 2 > . < add_3, 1 > . < wr_3, 2 >$
$N_{21} =< wr_1, 1 > || < wr_3, 2 >$
$N_{22} =< wr_2, 1 > || < wr_3, 2 >$
$N_{23} =< wr_3, 1 >$
$N_{24} =< add_3, 1 > . < wr_3, 2 >$
$N_{25} =< wr_3, 2 >$
$N_{26} = STOP$

En el grafo hemos marcado en color azul un camino mínimo desde el nodo inicial al nodo final obtenido mediante la ecuación 3.1 presentada en el apartado anterior. El peso de dicho camino es de 11, lo cual significa que el tiempo mínimo necesario para que el sistema evoluciones del estado inicial al estado final es de 11 unidades de tiempo, esto es el tiempo para realizar la suma completa de las dos matrices.

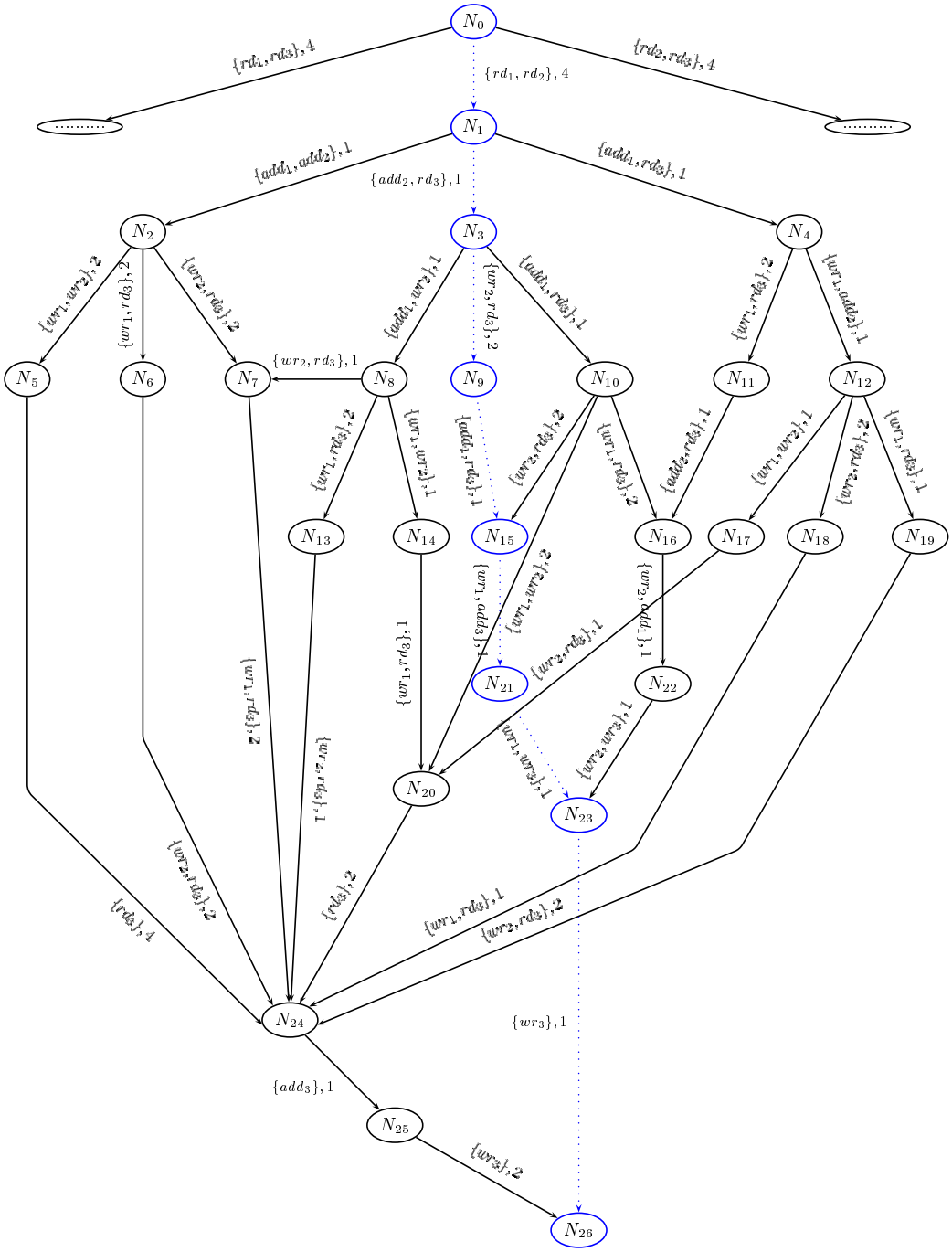


Figura 3.3: Grafo de transiciones para la suma de dos matrices 3x3 con $\mathcal{N} = 2$

Con este sencillo ejemplo ya se pueden observar importantes características de nuestro lenguaje. Nuestra álgebra es capaz de trabajar con paralelismo

real, para ello hemos modelado como recurso compartido los procesadores reales existentes en el sistema y somos capaces de modelar la ejecución simultanea de más de una acción (gracias a las bolsas de acciones). Un estudio de este mismo sistema se puede realizar considerando que sólo se cuenta con un procesador, con lo que a pesar de tener distintos procesos, realmente se estaría trabajando en interleaving. Este caso también se ha estudiado obteniendo un tiempo mínimo para la realización de la suma de las matrices de 21 unidades de tiempo.

Este mismo ejemplo de la suma de matrices se ha estudiado también para distintos tamaños de matrices y diferentes números de procesadores, obteniendo los resultados mostrados en la Tabla 3.6 donde se puede observar que nuestro modelo refleja satisfactoriamente el comportamiento esperado del sistema, disminuyendo el tiempo de procesamiento al aumentar el número de procesadores disponibles hasta llegar a un punto en el que cualquier aumento en el número de procesadores no conllevaría ninguna mejora o muy poca.

Hilos	Procesadores	T. Min	T. Max
3	2	11	21
3	3	7	21
4	2	14	28
4	3	10	28
4	4	7	28
6	2	21	42
6	3	14	42
6	4	11	42
6	6	7	42
10	2	35	70
10	3	24	70
10	4	18	70
10	5	14	70
10	6	12	70
10	7	11	70
10	8	11	70
10	9	11	70
10	10	7	70

Cuadro 3.6: Suma de dos matrices con distintos tamaños y diferentes números de procesadores

Con estos resultados estamos viendo los dos fines principales para los que puede ser usada nuestra álgebra; por un lado, partiendo de un número conocido de recursos disponibles, podemos describir el comportamiento y realizar un análisis temporal de éste. En este ejemplo podemos saber que para sumar dos matrices de 3x3 disponiendo de dos procesadores (recurso compartido), el tiempo mínimo necesario para realizar su suma será de 11 unidades de tiempo y el máximo nunca excederá de 21 unidades.

Por otro lado, si partimos de una especificación, este álgebra nos permite encontrar el número más adecuado de recursos para los que el comportamiento del sistema es el deseado. Esto es particularmente importante en los sistemas en los que el coste de los recursos es esencial. Para ello lo que hacemos es realizar el análisis de prestaciones para distintas configuraciones del sistema cambiando en cada una de ellas el número de recursos disponibles. Un estudio posterior de los resultados obtenidos servirá como apoyo a la hora de decidir cuál será la configuración que más se ajusta a nuestras necesidades. Por ejemplo, en la tabla 3.6 tenemos un caso claro. Queremos realizar la suma de dos matrices 10x10, para ello hacemos el análisis de prestaciones del sistema contando con 2 procesadores obteniendo que el tiempo mínimo necesario es de 35 unidades de tiempo lo cual supone una gran mejoría frente a las 70 unidades de tiempo necesarias si sólo contáramos con un procesador. Sin embargo vemos puntos donde la mejoría no es tan significativa, por ejemplo, el aumentar de 5 a 6 el número de procesadores conlleva una mejoría de tan sólo dos unidades de tiempo por lo que requerirá un estudio detallado por parte del diseñador del sistema de los beneficios de implantar esa opción. De gran ayuda, sin embargo, es este estudio para descubrir que el tiempo mínimo obtenidos con 7, 8 y 9 procesadores es el mismo, por lo que no tiene ninguna utilidad la inversión en el aumento de los recursos en este punto.

3.6. BTC con Recursos Heterogéneos

En este punto contamos con una primera versión de nuestra álgebra en la que principalmente nos centramos en poder trabajar con concurrencia real. Trabajamos con recursos homogéneos, esto es, somos capaces de tomar en consideración el número de recursos que en un momento determinado hay disponibles en el sistema pero todos esos recursos son de un mismo tipo. Con esto hemos añadido una mejora importante a las álgebras tradicionales ya que, como se ha comentado anteriormente, considerando como recurso compartido el procesador somos capaces de modelar tanto paralelismo real como interleaving.

Una vez definida esta primera versión, damos un paso más con el fin de modelar de una forma más fidedigna los sistemas reales. Encontramos que en los sistemas habitualmente existen más de un tipo de recursos compartido por los procesos, por ello, nosotros modificamos nuestra álgebra para que sea capaz de poder tomar en consideración distintos tipos de recursos compartidos (al mismo tiempo que controla cuántos están disponibles en cada momento). Esto es lo que nosotros llamamos *Recursos heterogéneos*. Además añadimos la posibilidad de que una acción necesite para su ejecución más de un recurso. Pasemos a ver como hemos llevado a cabo estas modificaciones.

Ahora, en un sistema podemos encontrar $m \in \mathbb{N}$ tipos diferentes de recursos compartidos. Para cada uno de estos tipos de recursos definimos un conjunto Z_i formado por todas las acciones que requieren para su ejecución al menos uno de los recursos compartidos del tipo i. La cardinalidad de este conjunto, esto es, el número de acciones que necesitan usar ese recurso será $x_i \in \mathbb{N}$. Así, cada conjunto Z_i se definirá de la siguiente manera:

$$Z_i = \{b_1,\ b_2,\ ...,\ b_{x_i}\}$$

Unimos todos estos conjuntos Z_i y definimos el conjunto Z del siguiente modo:

$$Z = \{Z_1,\ Z_2,\ ...,\ Z_m\}$$

donde vemos que Z está formado por todos los conjuntos Z_i generados para cada diferente tipo de recurso compartido existente en el sistema. Ahora consideremos

$$\mathcal{N} = \{N_1,\ N_2,\ ...,\ N_m\}$$

donde $N_i \in \mathbb{N}$ representa la cantidad de recursos compartidos del tipo i disponibles en el sistema.

Con todo esto, extendemos nuestra sintaxis para recoger esta información y poder saber que acciones necesitan de cada recurso y cuántos de ellos hay disponibles. Así, un proceso ahora se denotará como:

$$\{\![P]\!\}_{Z,\mathcal{N}}$$

Además, para poder trabajar con recursos heterogéneos necesitamos definir una nueva operación sobre multiconjuntos, en concreto lo que hacemos es redefinir la operación de cardinalidad de modo que $|B|_Z$ es el número de acciones del multiconjunto que necesitan para su ejecución de un recurso del conjunto Z. Esto es, $|B|_Z = \sum_{x \in Z} B(x)$.

Evidentemente, la semántica operacional también debe ser modificada y quedaría como se muestra en la Figura 3.7 donde cabe destacar que la regla $R5a$ captura el hecho de que el número máximo de acciones que no sean de sincronización que se pueden ejecutar simultáneamente viene delimitado por el número de recursos disponibles de cada tipo.

R1a) $$\frac{}{a.P \xrightarrow{\{a\},0} P}$$

R1b) $$\frac{}{< b, \alpha > .P \xrightarrow{\{b\},\alpha} P}$$

R1c) $$\frac{0 < \alpha' < \alpha}{< b, \alpha > .P \xrightarrow{\{b\},\alpha'} < b, \alpha - \alpha' > .P}$$

R2a) $$\frac{}{P \oplus Q \xrightarrow{\{\tau\},0} P}$$

R2b) $$\frac{}{P \oplus Q \xrightarrow{\{\tau\},0} Q}$$

R3a) $$\frac{P \xrightarrow{B,\alpha} P'}{P + Q \xrightarrow{B,\alpha} P'}$$

R3b) $$\frac{Q \xrightarrow{B,\alpha} Q'}{P + Q \xrightarrow{B,\alpha} Q'}$$

R4) $$\frac{P \xrightarrow{B',0} P' \wedge Q \xrightarrow{B',0} Q' \wedge B' = B' \cap A \neq \emptyset}{P \parallel_A Q \xrightarrow{B',0} P' \parallel_A Q'}$$

R5a) $$\frac{P \xrightarrow{B_1,\alpha} P' \wedge Q \xrightarrow{B_2,\alpha} Q' \wedge B_i \cap A = \emptyset \wedge \forall i \; |B_1|_{Z_i} + |B_2|_{Z_i} \leq N_i}{P \parallel_A Q \xrightarrow{B_1 \cup B_2,\alpha} P' \parallel_A Q'}$$

R5b) $$\frac{P \xrightarrow{B,\alpha} P' \wedge B \cap A = \emptyset}{P \parallel_A Q \xrightarrow{B,\alpha} P' \parallel_A Q}$$

R5c) $$\frac{Q \xrightarrow{B,\alpha} Q' \wedge B \cap A = \emptyset}{P \parallel_A Q \xrightarrow{B,\alpha} P \parallel_A Q'}$$

R6) $$\frac{P\{recX.P/X\} \xrightarrow{B,\alpha} P'}{recX.P \xrightarrow{B,\alpha} P'}$$

Cuadro 3.7: Semántica Operacional con Recursos Heterogéneos

3.7. Qué hemos mejorado

En este punto todavía se podía cuestionar si realmente era necesaria la creación de una nueva álgebra. Claramente nosotros creemos que si y pensamos que una buena demostración de ello sería compararla con un álgebra temporal de importante reconocimiento. Así, lo que vamos a mostrar a continuación es un ejemplo clásico de concurrencia modelado por **BTC** y por una reconocida extensión temporal de CSP, en concreto la desarrollada por Roscoe en [Ros98].

El sistema que queremos especificar es una Oficina de Correos. Los clientes que llegan a dicha oficina de correos para enviar sus giros postales siguen la siguiente conducta: en principio tienen que ir a un mostrador en el que se les proporciona un impreso, que a continuación tiene que cumplimentar. Para cumplimentar los impresos se dispone únicamente de 3 bolígrafos, dispuestos sobre la mesa (se supone que los clientes no llevan bolígrafos). Cuando un cliente acude a la mesa y no encuentra ninguno disponible, debe esperar a que algún otro cliente termine de utilizarlo y lo deposite sobre la mesa. Una vez cumplimentado el impreso, el cliente dispone de dos ventanillas (con una única cola para ambas) para el pago de la cantidad que desea enviar. Finalmente, una vez abonada la cantidad correspondiente se dirige a un único buzón, donde deposita su impreso, junto con el resguardo del pago.

Las dos álgebras que utilizaremos soportan características temporales por lo que a cada acción le asignaremos el tiempo necesario para su realización, el cuál, más tarde, será útil a la hora de hacer análisis de prestaciones.

Empecemos mostrando la especificación en **BTC**. Empezamos identificando los elementos del sistema. Claramente se aprecia que los clientes serán los procesos (users). Además en el sistema encontramos cuatro tipos distintos de recursos que deberán de ser compartidos por los distintos clientes que lleguen a esta oficina. El primer tipo de recurso es el mostrador donde se recoge el impreso que hay que rellenar. De este recurso el sistema sólo cuenta con una unidad. Otro tipo de recurso será el bolígrafo del cual encontramos tres unidades. El tercer tipo de recurso que identificamos es la ventanilla de pago. Disponemos de dos recursos de este tipo. Y finalmente contamos con un sólo recurso del último tipo: el buzón. Con esto podemos definir el conjunto $\mathcal{N}$ del siguiente modo:

$$\mathcal{N} = \{1, 3, 2, 1\}$$

Ahora debemos definir que acciones necesitan cada tipo de recurso para su ejecución. Como es un ejemplo sencillo tendremos una única acción que necesite cada tipo de recurso aunque evidentemente esto no es siempre así. Estas acciones serán: recoger el impreso ($pick\_up\_form_i$) que necesita del recurso mostrador (tipo 1), rellenar el impreso ($fill\_out\_form_i$) que requiere poder utilizar un bolígrafo (tipo 2), realizar el pago ($make\_payment_i$) que utilizará una ventanilla (tipo 3) y por último depositar el impreso ($place\_receipt$) que utiliza un recurso del tipo buzón (tipo 4). Con esto obtenemos los conjuntos Z_1 - Z_4:

$$Z_1 = \{pick\_up\_form_i\}$$
$$Z_2 = \{fill\_out\_form_i\}$$
$$Z_3 = \{make\_payment_i\}$$
$$Z_4 = \{place\_receipt_i\}$$

Como último paso para poder especificar el sistema nos quedaría definir la traza de cada proceso, en este caso de proceso $user_i$, pero como es trivial no lo haremos y pasamos directamente a mostrar como quedaría la especificación de este sistema utilizando el álgebra de procesos **BTC** (Figura 3.4).

$$Z_1 = \{pick\_up\_form_i\} \quad Z_2 = \{fill\_out\_form_i\}$$
$$Z_3 = \{make\_payment_i\} \quad Z_4 = \{place\_receipt_i\} \qquad \mathcal{N} = \{1,3,2,1\}$$
$$\{[\![\text{post\_office\_sys}]\!\}_{Z,\ \mathcal{N}} \equiv \{[\![user_1 \parallel \ldots \parallel user_i \parallel \ldots \parallel user_m]\!\}_{Z,\ \mathcal{N}}$$
$$user_i \equiv < pick\_up\_form_i, 2 > . < fill\_out\_form_i, 5 > .$$
$$< make\_payment_i, 3 > . < place\_receipt_i, 1 >$$

Figura 3.4: Especificación Oficina de Correos con **BTC**

Ahora pasemos a modelar este mismo sistema con el álgebra temporal de Roscoe. Lo primero que hacemos es identificar los elementos del sistema. Como procesos volvemos a identificar a los clientes ($user_i$), pero ahora nos encontramos con un problema habitual en las álgebras de procesos que intentan tener en cuenta la disponibilidad de los recursos del sistema; necesitan modelar estos recursos como procesos. Esto es una práctica bastante usada pero, evidentemente, no dará como resultado especificaciones muy intuitivas (los recursos son procesos) ni mucho menos sencillas. Veamos como queda la especificación antes de comentar algún otro aspecto de interés (Figura 3.5).

$$
\begin{aligned}
post\_office = {} & (user_1 \parallel \ldots \parallel user_i \parallel \ldots \parallel user_m) && \underset{\{form\_req, form\_res\}}{\parallel} \\
& form\_window && \underset{\{pen\_req, pen\_res\}}{\parallel} \\
& (pen\_table_1 \parallel pen\_table_2 \parallel pen\_table_2) && \underset{\{window\_req, window\_res\}}{\parallel} \\
& (pay\_window_1 \parallel pay\_window_2) && \underset{\{mailbox\_req, mailbox\_res\}}{\parallel} \\
& mailbox
\end{aligned}
$$

$$
\begin{aligned}
user_i = {} & form\_req \rightarrow pick\_up\_form \rightarrow tock \rightarrow tock \rightarrow form\_res \rightarrow \\
& pen\_req \rightarrow fill\_out\_form \rightarrow tock \rightarrow tock \rightarrow tock \rightarrow tock \rightarrow \\
& tock \rightarrow pen\_res \rightarrow window\_req \rightarrow make\_payment \rightarrow \\
& tock \rightarrow tock \rightarrow tock \rightarrow window\_res \rightarrow mailbox\_req \rightarrow \\
& place\_receipt \rightarrow tock \rightarrow mailbox\_res
\end{aligned}
$$

$$form\_window = form\_req \rightarrow form\_res$$

$$pen\_table_i = pen\_req \rightarrow pen\_res$$

$$pay\_window_i = window\_req \rightarrow window\_res$$

$$mailbox = mailbox\_req \rightarrow mailbox\_res$$

Figura 3.5: Especificación Oficina de Correos con la extensión temporal del álgebra CSP desarrollada por Roscoe

Otro punto interesante que podemos observar en esta última especificación es que el control de acceso a los distintos recursos debe realizarla el diseñador del sistema mientras que en la especificación con **BTC** no es necesario. Para explicar el por qué de este hecho podemos comparar el acceso a los recursos con el acceso a secciones críticas en la teoría de sistemas operativos. Así, podemos argumentar que **BTC** usa para acceder a los recursos un método similar al utilizado por los *Monitores* donde para acceder a un recurso en exclusión mutua sólo es necesario solicitar dicho recurso, siendo el sistema el encargado de garantizar que el número total de procesos usando un recurso (o instancia de recurso) no es mayor que el disponible. Por otro lado, el álgebra de Roscoe trabaja con un método similar al usado por los *Semáforos* donde para asegurar que el acceso a una sección crítica se realiza de una manera adecuada, se debe controlar por medio de la especificación.

Otro diferencia importante entre estas dos especificaciones se encuentra en el modo de asignar los recursos. En **BTC** realmente no se asigna un recurso a un proceso, esto es, un proceso que necesita un recurso de un cierto tipo y lo encuentra disponible lo utiliza durante el tiempo que le sea posible, pero si debe interrumpirse su utilización para más tarde continuarla, cuando el proceso reanude su ejecución puede optar nuevamente a *cualquier* recurso de ese mismo tipo, no necesariamente el mismo. En el álgebra de Roscoe, de igual modo que ocurre en casi todas las propuestas presentadas en teoría de scheduling donde se trata el tema de la asignación de recursos más en detalle, una vez que un proceso ha comenzado a utilizar un cierto recurso ya no tiene opción a utilizar otro de ese mismo tipo ya que aquí si se realiza asignación estática de recursos a procesos. Para apreciar mejor la importancia de este hecho basta con imaginar que el recurso en cuestión es de tipo procesador y fácilmente se adivina la evidente diferencia en el rendimiento que el sistema presentaría de tener una opción u otra.

Y para terminar sólo hacer notar la diferencia en tamaño y especialmente en sencillez entre ambas especificaciones. El sistema presentado es realmente tan sólo un ejemplo y tiene un tamaño realmente reducido pero en él ya se puede intuir que en sistemas reales, que siempre presentan un tamaño considerablemente mayor, esta diferencia será también más evidente.

3.8. BTC con Recursos Expropiativos y no Expropiativos

BTC en este punto es un álgebra capaz de especificar un amplio rango de sistemas reales de una manera bastante satisfactoria, pero se detecta algo que queda fuera de su alcance. Empecemos viendo un caso práctico. Intentamos modelar las acciones que realizan los mecánicos de un taller donde se cambian ruedas de coches y nos centramos principalmente en la utilización de los recursos. Cada mecánico que va a cambiar una rueda necesitará disponer de un gato hidráulico para tener en todo el proceso el coche elevado, una llave para aflojar y apretar los tornillos y una bomba de compresión. Al intentar modelar este sistema apreciamos que el recurso de tipo *gato* hidráulico debe ser utilizado durante todo el proceso del cambio de rueda. En principio, como nuestra álgebra es capaz de modelar acciones que utilicen más de un recurso, no parece que vayamos a tener problemas. El *gato* hidráulico es el tipo de recurso 1, la *llave* el tipo 2 y la *bomba* de compresión de tipo tres y en el sistema tenemos 4 *gatos*, 7 *llaves* y 2 *bombas*. Con esto podría hacerse un especificación como la que se muestra en la Figura 3.6.

$Z_1 = \{elevar\_c, bajar\_c, aflojar\_r, quitar\_rueda, poner\_rueda, apretar\_r, dar\_aire\}$
$Z_2 = \{aflojar\_r, apretar\_r\}$
$Z_3 = \{dar\_aire\}$

N= {2,4,7}

$\{[\text{sys\_taller}]\}_{Z,\,\mathcal{N}} \equiv \{[mecanico_1 \parallel \ldots \parallel mecanico_i \parallel \ldots \parallel mecanico_m]\}_{Z,\,\mathcal{N}}$

$mecanico_i \equiv < elevar\_c, 2 > . < aflojar\_r, 10 > . < quitar\_rueda, 5 > . < poner\_rueda, 5 > . < apretar\_r, 12 > . < dar\_aire, 7 > . < bajar\_coche, 2 >$

Figura 3.6: Especificación Taller Mecánico con **BTC**

Pero en esta especificación vemos un problema derivado directamente de una de las ventajas que hemos señalado que tenía nuestra álgebra: la no asignación de recursos. En esta especificación para poder ejecutarse, por ejemplo, la acción $aflojar_r$ se necesita un recurso de tipo *gato* y otro de tipo *llave*; hasta aquí ningún problema, se realiza la acción que tiene una duración de 10 unidades de tiempo y se liberan los dos recursos utilizados. Pero a continuación la acción que se debe de realizar es *quitar_rueda* para la que se vuelve a necesitar el recurso *gato*. El sistema comprobaría si hay recursos de ese tipo disponibles, y en caso de ser así se podría realizar la acción. Pero, realmente no nos vale cualquier recurso de tipo *gato*, es necesario que sea el mismo que se le asignó en la acción de $aflojar\_r$ que realizó ese mismo mecánico (proceso), es evidente que no se va a estar subiendo y bajando el coche con un gato distinto para cada acción. Para solucionar este problema la técnica más habitual que hemos encontrado se basa en especificar este tipo de recurso *gato* como un proceso con el que se debe sincronizar para simular su utilización. Nosotros hemos decidido modificar nuestro lenguaje para que estos recursos puedan ser modelados como tales sin necesidad de convertirlos en procesos.

Para ello vamos a diferenciar los recursos de dos tipo: expropiativos y no expropiativos. Un **recurso expropiativo** es uno que se puede arrebatar al proceso que lo tiene sin que haya efectos adversos. La memoria es un ejemplo de recurso expropiativo. Consideremos, por ejemplo, un sistema con 512K de memoria de usuario, una impresora y dos procesos de 512k que quieren imprimir algo. El proceso A solicita y obtiene la impresora, imprime algo y

comienza a calcular más valores que va a imprimir, pero antes de que haya terminado el cálculo excede su tiempo de procesador y es intercambiado a disco. Ahora se ejecuta el proceso B e intenta, sin éxito, adquirir la impresora. Aquí tenemos una situación problemática en potencia, porque A tiene la impresora y B tiene la memoria, y ninguno puede continuar sin el recurso que el otro tiene. Por fortuna, es posible quitarle la memoria a B (expropiarla) intercambiando B a disco y pasando A a la memoria. Ahora A puede ejecutarse, imprimir, y por último liberar la impresora.

En contraste, un **recurso no expropiativo** no puede quitársele a su poseedor actual sin generar efectos colaterales. Si un proceso ya comenzó a imprimir salidas y se le quita la impresora para dársela a otro proceso, se obtendrá basura como salida. Las impresoras no son expropiativos.

Nuestra álgebra debe ser modificada para ser capaz de diferenciar y tratar con estos dos tipos de recursos. La primera modificación importante que debemos hacer es considerar ahora tres tipos de acciones: **acciones temporales** (*timed actions*) que consumen tiempo y pueden usar recursos en caso de necesitarlos; las **acciones sin tiempo** (*untimed actions*) que se utilizarán para sincronizaciones y que no consumen tiempo ni recursos y las **acciones especiales** (*special actions*) que utilizan recursos pero no consumen tiempo y que utilizaremos para trabajar con los recursos no expropiativos. Con esto podemos definir nuevamente la sintaxis de **BTC** como sigue.

Sea $\mathcal{A}ct_T$ un conjunto finito de *acciones temporales*, $\mathcal{A}ct_U$ un conjunto finito de *acciones sin tiempo* y $\mathcal{A}ct_S$ un conjunto finito de *acciones especiales*, $\mathcal{A}ct_U \cap \mathcal{A}ct_T = \emptyset, \mathcal{A}ct_T \cap \mathcal{A}ct_S = \emptyset$ y $\mathcal{A}ct_U \cap \mathcal{A}ct_S = \emptyset$. La sintaxis de **BTC** queda definida por la siguiente expresión BNF:

$$P ::= stop \mid a.P \mid \langle b, \alpha \rangle.P \mid P \ \oplus \ P \mid P \ + \ P \mid P \parallel_A P \mid recX.P$$

donde $A \subseteq \mathcal{A}ct_U$, $a \in (\mathcal{A}ct_U \cup \mathcal{A}ct_S)$, $b \in \mathcal{A}ct_T$, y $\alpha \in \mathbb{N}$, $\mathbb{N}$ representa el conjunto de números naturales. Además, asumimos un conjunto de variables de procesos Id tomando valores en X, X', un conjunto de procesos $\mathcal{P}$ que coge sus valores en P, Q, R, y un conjunto de acciones $\mathcal{A}ct = \mathcal{A}ct_U \cup \mathcal{A}ct_T \cup \mathcal{A}ct_S$.

Para usar los recursos no expropiativos, en general, siempre se debe solicitar el recurso, utilizarlo y más tarde liberarlo. Veamos como modelamos esto con nuestro lenguaje.

Sea $c \in \mathcal{A}ct_S$ una *acción especial* que usaremos para solicitar un recurso y definimos su acción conjugada como $\widehat{c} \in \mathcal{A}ct_S$ que será utilizada para liberar dicho recurso.

Por supuesto nuestra álgebra sigue teniendo en cuenta el número de recursos de cada tipo (expropiativos y no expropiativos) que hay en el sistema. En el sistema encontramos $m \in \mathbb{N}$ tipos diferentes de recursos compartidos y para cada uno de estos tipos definimos un conjunto Z_i formado por todas las acciones que necesitan para su ejecución al menos un recurso de tipo i.

En el caso de que el recurso sea expropiativo, las acciones en el conjunto Z_i serán *acciones temporales* y denotamos por $x_i \in \mathbb{N}$ el número total de acciones en el conjunto Z_i. De este modo, cada conjunto Z_i para recursos expropiativos se define como:

$$Z_i = \{b_1,\ b_2,\ ...,\ b_{x_i}\}$$

Por otro lado, si los recursos son *no expropiativos*, en principio, sólo podemos encontrar dos acciones en su conjunto Z_i correspondiente. La acción c necesaria para solicitar el recurso y su conjugada $\widehat{c}$ para liberarlo. Por claridad, hemos decidido incluir en Z_i tan solo la acción c ya que se da por supuesto que siempre que aparezca una acción de este tipo aparecerá su conjugada. Así, para recursos *no expropiativos* el conjunto Z_i será:

$$Z_i = \{c_i\}$$

Juntamos estos dos tipos de conjuntos Z_i y definimos el conjunto Z:

$$Z = \{Z_1,\ Z_2,\ ...,\ Z_m\}$$

donde vemos que el conjunto Z está formado por todos los conjuntos Z_i generados para cada tipo de recurso. Además consideramos

$$\mathcal{N} = \{N_1,\ N_2,\ ...,\ N_m\}$$

donde $N_i \in \mathbb{N}$ representa la cantidad de recursos compartidos de tipo i disponibles en cada momento en el sistema. Y, para finalizar, recordamos que un proceso se denota de igual forma que en versiones anteriores de **BTC** por

$$\{\![P]\!\}_{Z,\mathcal{N}}$$

Veamos un simple ejemplo para entenderlo mejor. Trataremos de modelar un sistema responsable de pintar piezas. Un brazo articulado coge una pieza de una cinta transportadora y la lleva a una máquina (M_1) donde la pieza es limpiada (esta acción necesita dos unidades de tiempo para su realización);

además, en esta misma máquina la pieza pasará por el proceso de secado (cinco unidades de tiempo más). Después, el brazo articulado recoge la pieza y la transporta a una segunda máquina (M_2) donde es pintada, proceso que requiere tres unidades de tiempo.

En este sistema básico podemos observar que contamos tanto con recursos expropiativos como no expropiativos. Las dos máquinas M_1 y M_2 se modelarán con recursos expropiativos. Las acciones temporales *limpiar* y *secar* se ejecutarán en un recurso de tipo M_1 por lo que $Z_1 = \{limpiar, secar\}$ mientras que la acción *pintar* necesitara de un recurso de tipo M_2 con lo que $Z_2 = \{pintar\}$. En este sistema supondremos que contamos con cuatro máquinas de tipo M_1 y una de tipo M_2: $N_1 = 4$ y $N_2 = 1$.

Encontramos en el sistema otro tipo de recurso: el brazo articulado que es no expropiativo. Para cada nueva pieza que entra en el sistema se genera un nuevo proceso. Este nuevo proceso *pieza* solicitará un recurso del tipo brazo articulado (acción r_brazo) al comienzo de su ejecución y lo mantendrá hasta que finalice. El conjunto de acciones para este recurso serán r_brazo y su conjugado necesario para liberar el recurso: $\widehat{r\_brazo}$, pero como ya hemos dicho anteriormente, por motivos de claridad este conjugado no presentará en el conjunto Z_3 aún estando presente. Así, $Z_3 = \{r\_brazo\}$. Consideramos que en el sistema existen seis brazos articulados, esto es, seis recursos de tipo *brazo* por lo que N_3 valdrá 6. Con toda esta información la especificación del sistema quedaría como se muestra en la Figura 3.7

$$Z_1 = \{limpiar, secar\} \quad Z_2 = \{pintar\} \quad Z_3 = \{r\_brazo\}$$

$$Z = \{\{limpiar, secar\}, \{pintar\}, \{r\_brazo\}\}$$

$$\mathcal{N} = \{4, 1, 6\}$$

$$\{\![\text{Ex\_sys}]\!\}_{Z,\ \mathcal{N}} \equiv \{\![pieza_1 \parallel ... \parallel pieza_i]\!\}_{Z,\mathcal{N}}$$

$$pieza \equiv < r\_brazo, \mathcal{N} > .< limpiar, 2, \mathcal{N} > .< secar, 5, \mathcal{N} > .$$
$$< pintar, 3, \mathcal{N} > .< \widehat{r\_brazo}, \mathcal{N} >$$

Figura 3.7: Especificación Taller Pintura de piezas

Con esta este ejemplo sólo pretendemos dejar más claro como se construyen los conjuntos Z y $\mathcal{N}$ pero al hacer la especificación ya podemos observar más cambios que serán necesarios introducir en nuestra álgebra para poder ser capaz de tratar con recursos no expropiativos. Podemos observar que ca-

da acción ahora lleva asociado además de su tiempo de ejecución (en caso de que lo tenga) el valor $\mathcal{N}$ que, si recordamos, identificaba el número de recursos disponibles en el sistema. Veamos, en la siguiente sección, como necesitamos ampliar la semántica operacional para recoger esta información y que diferencia hay con la forma de recogerla en versiones anteriores de **BTC**.

3.8.1. Semántica operacional

En esta versión definitiva de **BTC** trabajamos con transiciones que son tripletas de la forma: $((P,\mathcal{N}),(Q,\mathcal{N}),(B,\alpha))$, y que nosotros habitualmente denotaremos por: $(P,\mathcal{N}) \xrightarrow{B,\alpha} (Q,\mathcal{N}')$.

Vemos una diferencia evidente con las versiones anteriores de **BTC** ya que ahora cada proceso lleva asociado el valor de $\mathcal{N}$, el cual recordemos que recoge en todo momento el número de recursos de cada tipo disponibles en el sistema. Esta información es necesaria para poder trabajar con recursos no expropiativos. En cualquier caso, el significado de la transición anterior es el habitual; el proceso P ejecuta las acciones del conjunto B durante un periodo de tiempo igual a α y después se comporta como el proceso Q.

El valor de $\mathcal{N}$ no se modifica a menos que un recurso no expropiativo sea solicitado o liberado. Después de esta afirmación parece lógico preguntarse por qué no se modifica cuando una acción temporizada está utilizando un recurso (expropiativo) ya que a fin de cuentas el número de recursos disponibles está variando. Esto es así por que cuando una acción necesita un recurso expropiativo para su ejecución, este recurso es utilizado (retenido) por un periodo igual al tiempo que dure la acción, por lo que al finalizar la acción ese recurso ha sido liberado y por lo tanto restablecido en $\mathcal{N}$. Esto se mantiene incluso en el caso de que una acción deba ser ejecutada en distintos momentos (esto es, si la acción es interrumpida) ya que los recursos expropiativos pueden ser expropiados a un proceso y reasignados más tarde (el mismo o cualquier otro recurso del mismo tipo) sin efectos secundarios.

En las Tablas 3.8, 3.9, 3.10, 3.11, y 3.12 podemos ver las reglas de la semántica operacional para la versión definitiva de **BTC** en las cuales suponemos que $\alpha,\alpha' \in \mathbb{N}$, $a \in \mathcal{A}ct_U$, $b \in \mathcal{A}ct_T$, $c \in \mathcal{A}ct_S$, $B_i \in \mathcal{B}(\mathcal{A}ct_T \cup \mathcal{A}ct_U\})$ y $B' \in \mathcal{B}(\mathcal{A}ct_U)$.

Vale la pena explicar algunas cosas que son bastantes diferentes de versiones anteriores. Empecemos por la regla $R1$ (Tabla 3.8) la cual muestra el funcionamiento del operador prefijo, pero que ha necesitado ser dividida para

R1a) $$\frac{}{(a.P, \mathcal{N}) \xrightarrow{\{a\},0} (P, \mathcal{N})}$$

R1b) $$\frac{}{(<b,\alpha>.P, \mathcal{N}) \xrightarrow{\{b\},\alpha} (P, \mathcal{N})}$$

R1c) $$\frac{0 < \alpha' < \alpha}{(<b,\alpha>.P, \mathcal{N}) \xrightarrow{\{b\},\alpha'} (<b,\alpha - \alpha'>.P, \mathcal{N})}$$

R1d) $$\frac{N_i > 0 \quad N_i \in \mathcal{N}}{(c_i.P, \mathcal{N}) \xrightarrow{\{c_i\},0} (P, \mathcal{N}')}$$

$$Donde\, \forall N_j \in \mathcal{N}\,, N'_j = \{ \begin{array}{ll} N_j & j \neq i \\ N_j - 1 & j = i \end{array}$$

R1e) $$\frac{}{(\hat{c}_i.P, \mathcal{N}) \xrightarrow{\{\hat{c}_i\},0} (P, \mathcal{N}')}$$

$$Donde\, \forall N_j \in \mathcal{N}\,, N'_j = \{ \begin{array}{ll} N_j & j \neq i \\ N_j + 1 & j = i \end{array}$$

Cuadro 3.8: Semántica Operacional Definitiva: Prefijo y Prefijo Temporizado

poder considerar los distintos tipos de acciones con los que ahora trabajamos. La regla $R1a$ es la clásica para el operador prefijo cuando la acción no utiliza tiempo ni recursos, en nuestro lenguaje para las *acciones sin tiempo*, pero aquí necesitamos incluir información adicional sobre la disponibilidad de los recursos en el sistema ($\mathcal{N}$). La regla $R1b$ establece que, dado un proceso P y una *acción temporizada* b con un tiempo de ejecución de α unidades, primero se ejecuta dicha acción b durante esas α unidades de tiempo y después el proceso se comporta como P.

En la regla $R1c$, como hemos visto en la sección anterior, se permite la ejecución parcial de una *acción temporizada*. La acción b se ejecuta durante α' unidades de tiempo dejando el resto $(\alpha - \alpha')$ para más tarde. Recordar que $\alpha \in \mathbb{N}$ por lo que esta regla no genera infinita transiciones al trabajar con un dominio discreto.

Finalmente, las reglas del operador prefijo para las *acciones especiales* son $R1d$ y $R1e$ que se han incluido nuevas. La primera es la utilizada cuando se solicita un recurso (no expropiativo) y la segunda para liberarlo. En la regla $R1d$, una acción especial que necesite un recurso de tipo i (c_i) para su ejecución sólo podrá ser llevada a cabo si en ese momento hay algún recurso de ese tipo disponible en el sistema ($N_i > 0$). La acción representa realmente el hecho de asignar un recurso por lo que después de ser ejecutada el conjunto $\mathcal{N}$ debe ser modificado y reflejar el hecho de que hay un recurso menos disponible de ese tipo en concreto. Cuando el recurso es liberado, la regla $R1e$ vuelve a modificar el valor de N_i pero ahora añadiéndole la unidad que se libera.

Como podemos ver en la Tabla 3.9, las reglas $R2$ y $R3$, que recogen el comportamiento del operador elección externa e interna respectivamente, no se modifican con respecto a la versión anterior.

R2a) $$\frac{}{(P,\mathcal{N}) \oplus (Q,\mathcal{N}) \xrightarrow{\{\tau\},0} (P,\mathcal{N})}$$

R2b) $$\frac{}{(P,\mathcal{N}) \oplus (Q,\mathcal{N}) \xrightarrow{\{\tau\},0} (Q,\mathcal{N})}$$

R3a) $$\frac{(P,\mathcal{N}) \xrightarrow{B,\alpha} (P',\mathcal{N}')}{(P,\mathcal{N}) + (Q,\mathcal{N}) \xrightarrow{B,\alpha} (P',\mathcal{N}')}$$

R3b) $$\frac{(Q,\mathcal{N}) \xrightarrow{B,\alpha} (Q',\mathcal{N}')}{(P,\mathcal{N}) + (Q,\mathcal{N}) \xrightarrow{B,\alpha} (Q',\mathcal{N}')}$$

Cuadro 3.9: Semántica Operacional Definitiva: Elección Interna y Externa

Para continuar con la presentación de esta versión definitiva de la semántica operacional de **BTC** recordar que la regla $R4$ (Tabla 3.10) recoge el mecanismo de sincronización donde sólo se tienen en consideración aquellas acciones que pueden ser ejecutadas tanto por el proceso P como por Q. Dado que todas las acciones de sincronización son *acciones sin tiempo*, el proce-

so completo evoluciona en 0 unidades de tiempo a $P' \parallel_A Q'$. La regla $R5a$ (Tabla 3.11) captura la ejecución simultanea de acciones que no son de sincronización con la restricción de que para cada tipo de recurso, se pueden estar ejecutando al mismo tiempo un máximo de N_i acciones y, finalmente, la regla $R6$ (Tabla 3.12) captura la semántica de la recursión.

R4) $$\frac{(P,\mathcal{N}) \xrightarrow{B',0} (P',\mathcal{N}) \wedge (Q,\mathcal{N}) \xrightarrow{B',0} (Q',\mathcal{N}) \wedge B' = B' \cap A \neq \emptyset}{(P,\mathcal{N}) \parallel_A (Q,\mathcal{N}) \xrightarrow{B',0} (P',\mathcal{N}) \parallel_A (Q',\mathcal{N})}$$

Cuadro 3.10: Semántica Operacional Definitiva: Sincronización

R5a) $$\frac{(P,\mathcal{N}) \xrightarrow{B_1,\alpha} (P',\mathcal{N}') \wedge (Q,\mathcal{N}) \xrightarrow{B_2,\alpha} (Q',\mathcal{N}') \wedge B_i \cap A = \emptyset \wedge \forall i\ |B_1|_{Z_i} + |B_2|_{Z_i} \leq N_i}{(P,\mathcal{N}) \parallel_A (Q,\mathcal{N}) \xrightarrow{B_1 \cup B_2,\alpha} (P',\mathcal{N}') \parallel_A (Q',\mathcal{N}')}$$

R5b) $$\frac{(P,\mathcal{N}) \xrightarrow{B,\alpha} (P',\mathcal{N}') \wedge B \cap A = \emptyset}{(P,\mathcal{N}) \parallel_A (Q,\mathcal{N}) \xrightarrow{B,\alpha} (P',\mathcal{N}') \parallel_A (Q,\mathcal{N})}$$

R5c) $$\frac{(Q,\mathcal{N}) \xrightarrow{B,\alpha} (Q',\mathcal{N}') \wedge B \cap A = \emptyset}{(P,\mathcal{N}) \parallel_A (Q,\mathcal{N}) \xrightarrow{B,\alpha} (P,\mathcal{N}) \parallel_A (Q',\mathcal{N}')}$$

Cuadro 3.11: Semántica Operacional Definitiva: Paralelismo

R6) $$\frac{(P\{recX.P/X\},\mathcal{N}) \xrightarrow{B,\alpha} (P',\mathcal{N}')}{(recX.P,\mathcal{N}) \xrightarrow{B,\alpha} (P',\mathcal{N}')}$$

Cuadro 3.12: Semántica Operacional Definitiva: Recursión

En la Tabla 3.13 podemos ver la versión completa y final de la semántica operacional de **BTC**.

R1a) $$\frac{}{(a.P,\mathcal{N}) \xrightarrow{\{a\},0} (P,\mathcal{N})}$$

R1b) $$\frac{}{(<b,\alpha>.P,\mathcal{N}) \xrightarrow{\{b\},\alpha} (P,\mathcal{N})}$$

R1c) $$\frac{0 < \alpha' < \alpha}{(<b,\alpha>.P,\mathcal{N}) \xrightarrow{\{b\},\alpha'} (<b,\alpha-\alpha'>.P,\mathcal{N})}$$

R1d) $$\frac{N_i > 0 \quad N_i \in \mathcal{N}}{(c_i.P,\mathcal{N}) \xrightarrow{\{c_i\},0} (P,\mathcal{N}')}$$
$$Where\, \forall N_j \in \mathcal{N}, N'_j = \begin{cases} N_j & j \neq i \\ N_j - 1 & j = i \end{cases}$$

R1e) $$\frac{}{(\hat{c}_i.P,\mathcal{N}) \xrightarrow{\{\hat{c}_i\},0} (P,\mathcal{N}')}$$
$$Where\, \forall N_j \in \mathcal{N}, N'_j = \begin{cases} N_j & j \neq i \\ N_j + 1 & j = i \end{cases}$$

R2a) $$\frac{}{(P,\mathcal{N}) \oplus (Q,\mathcal{N}) \xrightarrow{\{\tau\},0} (P,\mathcal{N})}$$

R2b) $$\frac{}{(P,\mathcal{N}) \oplus (Q,\mathcal{N}) \xrightarrow{\{\tau\},0} (Q,\mathcal{N})}$$

R3a) $$\frac{(P,\mathcal{N}) \xrightarrow{B,\alpha} (P',\mathcal{N}')}{(P,\mathcal{N}) + (Q,\mathcal{N}) \xrightarrow{B,\alpha} (P',\mathcal{N}')}$$

R3b) $$\frac{(Q,\mathcal{N}) \xrightarrow{B,\alpha} (Q',\mathcal{N}')}{(P,\mathcal{N}) + (Q,\mathcal{N}) \xrightarrow{B,\alpha} (Q',\mathcal{N}')}$$

R4) $$\frac{(P,\mathcal{N}) \xrightarrow{B',0} (P',\mathcal{N}) \wedge (Q,\mathcal{N}) \xrightarrow{B',0} (Q',\mathcal{N}) \wedge B' = B' \cap A \neq \emptyset}{(P,\mathcal{N}) \parallel_A (Q,\mathcal{N}) \xrightarrow{B',0} (P',\mathcal{N}) \parallel_A (Q',\mathcal{N})}$$

R5a) $$\frac{(P,\mathcal{N}) \xrightarrow{B_1,\alpha} (P',\mathcal{N}') \wedge (Q,\mathcal{N}) \xrightarrow{B_2,\alpha} (Q',\mathcal{N}') \wedge B_i \cap A = \emptyset \wedge \forall i\ |B_1|_{Z_i} + |B_2|_{Z_i} \leq N_i}{(P,\mathcal{N}) \parallel_A (Q,\mathcal{N}) \xrightarrow{B_1 \cup B_2,\alpha} (P',\mathcal{N}') \parallel_A (Q',\mathcal{N}')}$$

R5b) $$\frac{(P,\mathcal{N}) \xrightarrow{B,\alpha} (P',\mathcal{N}') \wedge B \cap A = \emptyset}{(P,\mathcal{N}) \parallel_A (Q,\mathcal{N}) \xrightarrow{B,\alpha} (P',\mathcal{N}') \parallel_A (Q,\mathcal{N})}$$

R5c) $$\frac{(Q,\mathcal{N}) \xrightarrow{B,\alpha} (Q',\mathcal{N}') \wedge B \cap A = \emptyset}{(P,\mathcal{N}) \parallel_A (Q,\mathcal{N}) \xrightarrow{B,\alpha} (P,\mathcal{N}) \parallel_A (Q',\mathcal{N}')}$$

R6) $$\frac{(P\{recX.P/X\},\mathcal{N}) \xrightarrow{B,\alpha} (P',\mathcal{N}')}{(recX.P,\mathcal{N}) \xrightarrow{B,\alpha} (P',\mathcal{N}')}$$

Cuadro 3.13: Semántica Operacional Definitiva: con recursos Heterogéneos, Expropiativos y no Expropiativos

3.8.2. Ejemplo: Comedor

En esta sección veremos un ejemplo sencillo para tener una idea clara de como **BTC** trabaja tanto con recursos expropiativos como no expropiativos. En capítulos posteriores se estudiaran ejemplos más complejos y más próximos a la realidad, pero este es una buena muestra de la sencillez de las especificaciones obtenidas y de la gran potencia de nuestro lenguaje.

Vamos a intentar modelar un comedor al que acceden unos clientes con el objetivo de comer. Cuando un cliente entra en el establecimiento, lo primero

que tiene que hacer es recoger una bandeja (se cuenta con 25 bandejas) e ir a por su comida que es servida por tres camareros. Una vez recogida la comida el cliente se la come y paga la cuenta (sólo hay una caja de pago).

Observamos que tenemos un recurso que debe mantenerse durante la realización de varias acciones que además necesitan de otros recursos: el recurso *bandeja*. Es un caso similar al del recurso *gato* en el ejemplo anterior; el proceso *user* necesita mantener el mismo recurso *bandeja*, no le sirve obtener cualquier recurso de tipo *bandeja* cuando se dispone a comer, quiere que la bandeja sea la misma en la que realizó la acción de obtener la comida. Evidentemente el recurso bandeja es de tipo no expropiativo y será necesario solicitarlo para su utilización y liberarlo al acabar. Esto lo haremos con las acciones re_tray_i y su conjugado. Además encontramos otros recursos que si son expropiativos; los *camareros* y el *cajero*.

Suponiendo que la acción de coger la comida (get_food_i) necesita 5 unidades de tiempo para realización, para comer (acción eat_i) 15 y para pagar (acción pay_food_i) 3 unidades más, la especificación del sistema quedaría como se muestra en la Figura 3.8.

$Z_1 = \{re\_tray_i\}$
$Z_2 = \{get\_food_i\}$ N= {25,3,1}
$Z_3 = \{pay\_food_i\}$

$$\{[\text{sys\_dining\_hall}]\}_{Z,\,\mathcal{N}} \equiv \{[user_1 \parallel \ldots \parallel user_i \parallel \ldots \parallel user_m]\}_{Z,\,\mathcal{N}}$$

$$user_i \equiv < re\_tray_i, \mathcal{N} > . < (get\_food_i, 5), \mathcal{N} > . < (eat_i, 15), \mathcal{N} > . < \widehat{re\_tray}_i, \mathcal{N} > . < (pay\_food_i, 3), \mathcal{N} >$$

Figura 3.8: Especificación del Comedor con Recursos expropiativos

Vemos que tenemos acciones especiales que consumen recursos pero no consumen tiempo: re_tray_i y su conjugado $\widehat{re\_tray}_i$. El recurso que consumen es de tipo bandeja y es decrementado del número de recursos disponibles de ese tipo al solicitarlo (re_tray_i) por la regla $R1d$. Más tarde volverá a quedar disponible al liberarlo ($\widehat{re\_tray}_i$), incrementando nuevamente el valor de N_1 por la regla $R1e$. Estas acciones, como vemos, no llevan tiempo asociado ya que no es posible saber cuando será el tiempo invertido en obtener los otros recursos necesarios para realizar las acciones; en este caso no se puede estimar cuánto tiempo necesitarás para que un camarero esté disponible para

servirte la comida. También encontramos una acción que sólo necesita de un recurso de tipo no expropiativo (eat_i) y otra que sólo necesita uno de tipo expropiativo (pay_food_i).

Una buena manera de ayudar a apreciar la gran potencia que nos aporta el hecho de poder distinguir entre recursos expropiativos y no expropiativos (y, consecuentemente, modelarlos de forma diferente) es comparar esta especificación del comedor que cuenta con recursos no expropiativos con la especificación que obtendríamos si no dispusiéramos de ellos (o se trataran de igual modo).

Evidentemente, el problema se presenta a la hora de modelar el recurso *bandeja*. Este recurso se requiere para poder ejecutar diversas acciones, en concreto será necesario para poder llevar a cabo las acciones get_food_i y eat_i. En principio esto no debe ser un problema ya que **BTC** permite modelar acciones que requieran del uso de más de un recurso al mismo tiempo. En este supuesto, los conjunto Z_i se modificarían quedando definidos como:

$$Z_1 = \{get\_food_i, eat_i\}$$
$$Z_2 = \{get\_food_i\}$$
$$Z_3 = \{pay\_food_i\}$$

para los recursos *bandeja*, *camarero* y *cajero* respectivamente. Pero aún así tenemos un problema importante: una vez finalizada la acción get_food_i, los recursos necesarios para su ejecución son liberados. Esto quiere decir que el recurso $bandeja_i$ que teníamos asignado se libera, y no tiene por que ser la misma instancia de dicho recurso la que se asigne cuando ese mismo usuario quiera ejecutar la acción eat_i. Es de suponer que el usuario desearía comer (ejecutar la acción eat_i) en la misma bandeja donde cogió la comida (get_food_i), pero aquí no tenemos forma de garantizarlo.

La alternativa que obtenemos es la misma que usan el resto de álgebras de procesos: crear un proceso para poder modelar el recurso *bandeja*. Para ello, creamos un proceso *despachador* de bandejas con el que el usuario necesitará sincronizar para simular la recogida ($tray\_req_i$) y la devolución ($tray\_ret_i$) de esa bandeja. La especificación que obtenemos de este modo es la mostrada en la Figura 3.9 donde podemos apreciar que, además de necesitar nuevos procesos, el resultado obtenido no es tan intuitivo como el anterior y la especificación aumenta de tamaño y complejidad considerablemente, lo cual para sistemas reales de mayor envergadura puede convertirse en un verdadero problema.

$Z_1 = \{get\_food_i\}$ N= {3,1}
$Z_2 = \{pay\_food_i\}$

$$\{[\text{sys\_dining\_hall}]\}_{Z,\ N} \equiv \{[user_{x(1)} \parallel \ldots \parallel user_{x(i)} \parallel \ldots \parallel user_{x(m)} \parallel tray\_dispatcher]\}_{Z,\ N}$$

$$user_{x(i)} \equiv tray\_req_{x(i)}. < get\_food_{x(i)}, 5 > . < eat_{x(i)}, 15 > .tray\_ret_{x(i)}. < pay\_food_{x(i)}, 3 >$$

$$tray\_dispatcher \equiv EB_1 \parallel EB_2 \parallel \ldots \parallel EB_i \parallel \ldots \parallel EB_{25}$$

$$EB_i \equiv tray\_req_i.tray\_ret_i.EB_i$$

Donde $\forall i \;\; x(i) = i \; MOD \; 25$

Figura 3.9: Especificación del Comedor sin Recursos expropiativos

Capítulo 4

Aplicación: Protocolo SET

Internet se ha convertido en las últimas décadas en una de las tecnologías más dominantes en el campo de los computadores. Tiene un gran impacto en el trabajo, el ocio y el conocimiento gracias a que da la posibilidad de comunicar personas o entidades, compartir información o recursos, permite acceso a educación remota o realizar compras on-line entre otras muchas cosas. Toda esta revolución generada por Internet ha traído consigo la aparición de una gama de nuevas tecnologías. En este capítulo, nosotros nos centraremos en una de ellas, la tecnología que ha permitido redefinir la manera en que se hacían negocios tradicionalmente; hablamos del Comercio Electrónico (E-Commerce o E-business). A modo de definición diremos que el Comercio Electrónico consiste principalmente en la distribución, compra, venta, marketing y suministro de información, artículos o servicios a través de redes informáticas como Internet.

La mayoría de las transacciones llevadas a cabo a través del Comercio Electrónico conlleva un tráfico de dinero de una manera u otra. El modo más común es utilizar tarjetas de crédito, pero sea cual sea el modo utilizado, lo cierto es que el pago debe de hacerse a través de una red que por definición es no segura. Al ser insegura, evidentemente existe la posibilidad de que maliciosos intrusos puedan acceder a ella y conseguir datos confidenciales de importancia como por ejemplo el número de una tarjeta de crédito, con lo que ello conllevaría. Esto crea la necesidad de tomar medidas de seguridad para el tráfico por la red, ya sea encriptando datos, identificando y autorizando a los participantes en las comunicaciones, limitando el acceso a recursos o implementado protocolos de seguridad.

Los intentos por conseguir que la transmisión de cualquier tipo de información a través de la red se realize de una manera segura son muchas, Secure Sockets Layer (SSL), Transport Layer Security (TLS) y Secure Electronic Transaction (SET) son probablemente los protocolos de seguridad más populares actualmente. El protocolo SSL es un sistema diseñado y propuesto por Netscape Communications Corporation con el fin de permitir la transmisión de documentos o datos vía Internet de una manera segura. SSL Proporciona cifrado de datos, autentificación de servidores, integridad de mensajes y, opcionalmente, autentificación de clientes para conexiones TCP/IP. El protocolo TLS es una evolución del protocolo SSL por lo que mantiene una compatibilidad total con él y que además permite que la seguridad pase del nivel de conexión (SSL) al nivel de transporte (TLS).

El protocolo SET es el último de los citados en aparecer y es un protocolo transaccional orientado a las aplicaciones de comercio electrónico que utilizan como medio de pago tarjetas de crédito. A diferencia de los otros protocolos de seguridad, que son de carácter general, SET fue expresamente diseñado para el comercio electrónico, por eso no supone ninguna mejora en cuanto a la seguridad en la comunicación propiamente dicha, pero mejora las condiciones en las que el proceso de comercio electrónico tiene lugar.

Todos los protocolos de seguridad tienen el objetivo común de garantizar una seguridad, y sus usuarios dependen en gran medida de la corrección de estos protocolos, ya que un mínimo fallo en ellos estaría comprometiendo la seguridad de los datos que intenta proteger. Por ello es doblemente importante ser capaces de probar la corrección de esos protocolos.

Ha habido muchos intentos de modelar y analizar formalmente dichos protocolos, esto es, probar su corrección o identificar posibles ataques a los que serían vulnerables. El primer trabajo que utilizó un método formal para este fin fue realizado por Dolev y Yao [DY83], quienes desarrollaron un conjunto de algoritmos con tiempo polinomial que se usaban para analizar la seguridad de una restringida clase de protocolos. Este trabajo tuvo una importancia excepcional ya que fue el primero que utilizando métodos formales consiguió modelar un entorno en el que multiples ejecuciones del protocolo podían ejecutarse de forma concurrente. De hecho, bastantes trabajos realizados más tarde con el mismo fin de analizar protocolos de seguridad, se basaban en este modelo ([MCF87], [LR92]).

Un trabajo que no podemos dejar de mencionar es el realizado por Lowe en [Low96], donde haciendo uso de un verificador de modelos de propósito general, basado en CSP, consiguió encontrar un ataqué al protocolo de Needham-Shroeder.

Centrandonos en los protocolos que se han utilizado concretamente para comercio electrónico, a partir del trabajo de Kailar [Kai95] en el que se analiza un sencillo protocolo de comercio electrónico, encontramos otros trabajos muy interesante basado en Kerberos [BP98], el Internet Key Exchange protocol [Mea99], Cybercash [LH01], SSL (Secure Sockets Layer) [MSS98] o su sucesor TLS [Pau98] y, por supuesto, basados en el protocolo más novedoso: SET el cual está siendo especialmente resistente a su verificación. Esta dificultad para su análisis radica principalmente en el hecho de que los protocolos de seguridad cada vez son más complejos, prueba de ello es que fueron suficientes seis páginas para describir el protocolo de Needham-Schroeder en 1978 mientras que con más de seiscientas páginas en la especificación del protocolo SET parecen no haber sido suficientes. También influye el hecho de que actualmente los protocolos de seguridad (y en concreto el protocolo SET) son realmente un "conjunto" de protocolos ya que claramente se pueden dividir en distintas fases con sus propias características y objetivos.

Entre los intentos de verificar este protocolo formalmente podemos destacar los trabajos de Paulson y Bella [BMP02, BMP03] que han conseguido modelar e incluso llegar a verificar distintas fases de una versión simplificada (aún así compleja) del protocolo SET mediante una metodología inductiva y haciendo uso del demostrador de teoremas Isabelle. Meadows y Syverson [MS98] han conseguido diseñar un lenguaje temporal, NPATRL (NRL Protocol Analyzer Temporal Requirements Language), para especificar una serie de requerimientos del protocolo SET pero por el momento han pospuesto su análisis para más adelante. Kessler y Neumann [KN98] han extendido una lógica existente con predicados y reglas que les permita razonar sobre responsabilidades (accountability) que, a pesar de no encontrarse entre los objetivos principales del protocolo SET siempre es deseable. Stoller [Sto01] también ha propuesto un marco formal para realizar un análisis de protocolos de comercio electrónico, pero sólo ha considerado una versión demasiado simplificada de la fase de pago del protocolo SET.

Con respecto a los trabajos encontrados sobre la utilización de Model Checking sobre especificaciones del protocolo SET destacamos los de Heintze [HTWW96] y Heitmeyer [HKL$^+$98] pero no entraremos en más detalle con ellos ya que se apartan un poco de nuestro trabajo.

Según nuestra opinión, queda otra importante línea de estudio formal que no ha sido tratada en el análisis del protocolo SET; el análisis de prestaciones. El protocolo SET es robusto, pero a costa de ser pesado y muy costoso en el ancho de banda requerido, y en tiempo tanto de transmisión como de cómputo criptográfico. Esto nos lleva a pensar que un análisis de presta-

ciones en el que se pueda modelar los tiempos requeridos para las distintas operaciones llevadas a cabo con dicho protocolo puede ser de gran utilidad o incluso necesidad. Con esta premisa nos decidimos a abordar dicho análisis de prestaciones haciendo uso de nuestra álgebra de procesos **BTC**. Por supuesto encontramos problemas similares a los encontrados en los intentos de realizar verificación o model checking sobre SET; es un protocolo demasiado complejo para ser especificado en su globalidad, pero después de un estudio detallado de su documentación logramos especificarlo con las mínimas simplificaciones posibles lo que nos permite realizar nuestro análisis de prestaciones.

4.1. Qué es SET

Internet promete nuevas oportunidades de negocio, pero plantea también importantes desafíos de seguridad que frenan la expansión del comercio electrónico entre empresas y consumidores. La llegada de nuevas soluciones de pago como SET pavimentan el camino hacia un comercio seguro a través de Internet.

En el panorama actual de Internet existen dos tipos de negocios: los que se sirven de la Red como de un mero escaparate virtual en el que anunciar sus productos y servicios, y los que utilizan mecanismos web para, además, vender esos productos. Por supuesto, la ventaja competitiva de estas empresas es muy superior a la de las primeras: su comercio, no sólo su catálogo, está disponible las 24 horas del día, siete días a la semana, accesible a un mercado potencialmente mundial. En contrapartida, también la complejidad de gestión de pagos a la que se enfrentan es mucho mayor.

Como cualquier otro canal de distribución, la Red plantea un conjunto único de retos de seguridad que deben afrontarse racionalmente para minimizar el riesgo y ofrecer una confianza sólida a los actores de las relaciones telemáticas que no se conocen entre ellos, tanto en el business-to-business entre empresas, como en el comercio al por menor entre vendedores y compradores particulares.

El protocolo SET para pagos seguros con tarjeta de crédito a través de Internet ofrece una solución para reducir costes de operación para el vendedor, aumentar la seguridad frente a otras tecnologías más extendidas como SSL a la vez que reduce el fraude y expande las fronteras de negocio de los comerciantes hacia nuevos mercados globales.

4.1.1. Por qué SET

¿Por qué la necesidad de protocolos como SET? ¿No bastaría con enviar los datos de la tarjeta de crédito a través de un formulario web? Al fin y al cabo miles de compras se realizan diariamente suministrando por teléfono esta información. ¿Dónde está la diferencia?

Lo cierto es que Internet presenta un obstáculo psicológico para la mayoría de los compradores, que inhibe el crecimiento de la Web como escenario de comercio electrónico. El temor al fraude con tarjetas de crédito retrae por igual a consumidores (que quieren estar seguros de comprar en una tienda real y que el comerciante no utilizará ilícitamente su tarjeta de crédito) y comerciantes (que de hecho se encuentran más desprotegidos que los clientes, ya que deben asumir los costes de transacciones ilícitas si más tarde el titular legítimo de la tarjeta rechaza el pedido), y este temor afecta tanto a los intereses económicos de los comerciantes, que ven reducidas sus ventas, como de las instituciones financieras, que ven disminuir sus comisiones por pagos con tarjeta.

El protocolo SSL [FKK96], actualmente mucho más extendido en Internet que SET, no fue diseñado para interacciones entre múltiples partes, como las transacciones comerciales, que pueden llegar a involucrar hasta seis partes. SSL se limita a cifrar el número de tarjeta de crédito cuando es transmitido desde el navegador del cliente hasta el servidor del comerciante, resultando insuficiente para los requisitos de seguridad de un comercio electrónico fiable.

Esta situación no se podía mantener por más tiempo, por lo que en 1995 Visa y MasterCard, con la colaboración de otras compañías líderes en el mercado de las tecnologías de la información, como Microsoft, IBM, Netscape, RSA, o VeriSign, unieron sus fuerzas para desarrollar Secure Electronic Transaction (SET), un protocolo estandarizado y respaldado por la industria, diseñado para salvaguardar las compras pagadas con tarjeta a través de redes abiertas, incluyendo Internet.

4.1.2. Cómo SET protege sus transacciones

El protocolo SET ofrece una serie de servicios que convierten las transacciones a través de Internet en un proceso seguro y fiable para todas la partes implicadas:

- Autentificación: todas las partes involucradas en la transacción económica (el cliente, el comerciante y los bancos, emisor y adquiriente) pueden verificar mutuamente sus identidades mediante certificados digitales. De esta forma, el comerciante puede asegurarse de la identidad del titular de la tarjeta y el cliente, de la identidad del comerciante. Se evitan así fraudes debidos a usos ilícitos de tarjetas y a falsificaciones de comercios en Internet (*web spoofing*), que imitan grandes web comerciales. Por su parte, los bancos pueden asimismo comprobar la identidad del titular y del comerciante.
- Confidencialidad: la información de pago se cifra para que no pueda ser espiada mientras viaja por las redes de comunicaciones. Solamente el número de tarjeta de crédito es cifrado por SET, de manera que ni siquiera el comerciante llegará a verlo, para prevenir fraudes. Si se quiere cifrar el resto de datos de la compra, como por ejemplo qué artículos se han comprado o a qué dirección deben enviarse, debe recurrirse a un protocolo de nivel inferior como SSL.
- Integridad: garantiza que la información intercambiada, como el número de tarjeta, no podrá ser alterada de manera accidental o maliciosa durante su transporte a través de redes telemáticas. Para lograrlo se utilizan algoritmos de firma digital, capaces de detectar el cambio de un solo bit.
- Intimidad: el banco emisor de la tarjeta de crédito no puede acceder a información sobre los pedidos del titular, por lo que queda incapacitado para elaborar perfiles de hábitos de compra de sus clientes.
- Verificación inmediata: proporciona al comerciante una verificación inmediata, antes de completarse la compra, de la disponibilidad de crédito y de la identidad del cliente. De esta forma, el comerciante puede cumplimentar los pedidos sin riesgo de que posteriormente se invalide la transacción.
- No repudio para resolución de disputas: la mayor ventaja de SET frente a otros sistemas seguros es la adición al estándar de certificados digitales (X.509v3), que asocian la identidad del titular y del comerciante con entidades financieras y los sistemas de pago de Visa, MasterCard, etc. Estos certificados previenen fraudes para los que otros sistemas no ofrecen protección, como el repudio de una transacción (negar que uno realizó tal transacción), proporcionando a los compradores y vendedores la misma confianza que las compras convencionales usando las actuales redes de autorización de créditos de las compañías de tarjetas de pago.

4.1.3. Quiénes participan en SET

El pago mediante tarjeta es un proceso complejo en el cual se ven implicadas varias entidades:

- El banco emisor: emite la tarjeta del cliente, extiende su crédito y es responsable de la facturación, recolección y servicio al consumidor. En el artículo 46 de la Ley de Comercio Minorista se establece que cuando el importe de una compra hubiese sido cargado utilizando el número de una tarjeta de crédito, sin que ésta hubiese sido presentada directamente o identificada electrónicamente (por ejemplo por un hacker que robó el número en Internet), su titular podrá exigir la inmediata anulación del cargo.
- El banco adquiriente: forma relación con el comerciante, procesando las transacciones con tarjeta y las autorizaciones de pago.
- El titular de la tarjeta: posee la tarjeta emitida por el banco emisor y realiza y paga las compras.
- El comerciante: vende productos, servicios o información y acepta el pago electrónico. La parte débil en las transacciones electrónicas es el comerciante, a quien corresponde probar que su abono está justificado (a no ser que responda el banco o entidad financiera titular de la tarjeta, todo depende del contrato que tenga con el comerciante).
- La pasarela de pagos: mecanismo mediante el cual se autorizan y procesan las transacciones del comerciante (autorización, revocación, liquidación, etc.). La pasarela puede pertenecer a una entidad financiera (adquiriente) o a un operador de medios de pago. Conectan Internet con las redes privadas de autorización de pagos, largamente establecidas y fiables.
- El procesador (redes de medios de pago): proporciona servicios adicionales operando la infraestructura de telecomunicaciones sobre las que se realizan las transacciones.
- Autoridad de certificación: certifica las claves públicas del titular de la tarjeta, del comerciante y de los bancos. La Agencia de Certificación Española (ACE), formada por Telefónica, SERMEPA, CECA y Sistema 4B, viene ofreciendo el servicio de certificación SET desde finales de 1998 en España, http://www.ace.es/.

4.1.4. Cómo funciona SET

El proceso subyacente en una transacción SET típica funciona de forma muy parecida a una transacción convencional con tarjeta de crédito:

1. Decisión de compra del cliente. El cliente está navegando por el sitio web del comerciante y decide comprar un artículo. Para ello rellenará algún formulario al efecto y posiblemente hará uso de alguna aplicación tipo carrito de la compra, para ir almacenando diversos artículos y pagarlos todos al final. El protocolo SET se inicia cuando el comprador pulsa el botón de Pagar o equivalente.

2. Arranque de la cartera. El servidor del comerciante envía una descripción del pedido que despierta a la aplicación cartera del cliente.

3. Transmisión cifrada de la orden de pago. El cliente comprueba el pedido y transmite una orden de pago de vuelta al comerciante. La aplicación *cartera* crea dos mensajes que envía al comerciante. El primero, la información del pedido, contiene los datos del pedido, mientras que el segundo contiene las instrucciones de pago del cliente (número de tarjeta de crédito, banco emisor, etc.) para el banco adquiriente.

   En este momento, el software cartera del cliente genera un firma dual, que permite juntar en un solo mensaje la información del pedido y las instrucciones de pago, de manera que el comerciante puede acceder a la información del pedido, pero no a las instrucciones de pago, mientras que el banco puede acceder a las instrucciones de pago, pero no a la información del pedido.

   Este mecanismo reduce el riesgo de fraude y abuso, ya que ni el comerciante llega a conocer el número de tarjeta de crédito empleado por el comprador, ni el banco se entera de los hábitos de compra de su cliente.

4. Envío de la petición de pago al banco del comerciante. El software SET en el servidor del comerciante crea una petición de autorización que envía a la pasarela de pagos, incluyendo el importe a ser autorizado, el identificador de la transacción y otra información relevante acerca de la misma, todo ello convenientemente cifrado y firmado. Entonces se envían al banco adquiriente la petición de autorización junto con las instrucciones de pago (que el comerciante no puede examinar, ya que van cifradas con la clave pública del adquiriente).

5. Validación del cliente y del comerciante por el banco adquiriente. El banco del comerciante descifra y verifica la petición de autorización. Si el proceso tiene éxito, obtiene a continuación las instrucciones de pago del cliente, que verifica a su vez, para asegurarse de la identidad del titular de la tarjeta y de la integridad de los datos. Se comprueban los identificadores de la transacción en curso (el enviado por el comerciante y el codificado en las instrucciones de pago) y, si todo es correcto, se formatea y envía una petición de autorización al banco emisor del cliente a través de la red de medios de pago convencional.

6. Autorización del pago por el banco emisor del cliente. El banco emisor verifica todos los datos de la petición y si todo está en orden y el titular de la tarjeta posee crédito, autoriza la transacción.

7. Envío al comerciante de un testigo de transferencia de fondos. En cuanto el banco del comerciante recibe una respuesta de autorización del banco emisor, genera y firma digitalmente un mensaje de respuesta de autorización que envía a la pasarela de pagos, convenientemente cifrada, la cual se la hace llegar al comerciante.

8. Envío de un recibo a la cartera del cliente. Cuando el comerciante recibe la respuesta de autorización de su banco, verifica las firmas digitales y la información para asegurarse de que todo está en orden. El software del servidor almacena la autorización y el testigo de transferencia de fondos. A continuación completa el procesamiento del pedido del titular de la tarjeta, enviando la mercancía o suministrando los servicios pagados. Además, se le entrega a la aplicación cartera del cliente un recibo de la compra para su propio control de gastos y como justificante de compra.

9. Entrega del testigo de transferencia de fondos para cobrar el importe de la transacción. Después de haber completado el procesamiento del pedido del titular de la tarjeta, el software del comerciante genera una petición de transferencia a su banco, confirmando la realización con éxito de la venta. Como consecuencia, se produce el abono en la cuenta del comerciante.

10. Cargo en la cuenta del cliente. A su debido tiempo, la transacción se hace efectiva sobre la cuenta corriente del cliente.

El protocolo definido por SET especifica el formato de los mensajes, las codificaciones y las operaciones criptográficas que deben usarse. No requiere un método particular de transporte, de manera que los mensajes SET pueden transportarse sobre HTTP en aplicaciones Web, sobre correo electrónico o cualquier otro método. Como los mensajes no necesitan transmitirse en tiempo real, son posibles implantaciones de SET eficientes basadas en correo electrónico u otros sistemas asíncronos.

En su estado actual SET solamente soporta transacciones con tarjeta de crédito/débito, y no con tarjetas monedero. Se está trabajando en esta línea para extender el estándar de manera que acepte nuevas formas de pago. Al mismo tiempo se están desarrollando proyectos para incluir los certificados SET en las tarjetas inteligentes, de tal forma que el futuro cambio de tarjetas de crédito a tarjetas inteligentes pueda incorporar el estándar SET.

4.1.5. Certificados digitales en SET

SET proporciona los mecanismos necesarios para que tanto consumidores como comerciantes se autentifiquen mutuamente antes de que la transacción tenga lugar. De esta manera se consigue replicar en el mundo digital la situación común en la que el cliente se encuentra físicamente delante del mostrador del vendedor a la hora de pagar la compra.

SET utiliza certificados digitales para realizar este proceso de autentificación. Estos certificados sirven como documentos de identidad digitales (algo así como un DNI virtual) que permiten verificar la identidad de una persona a través de una red de telecomunicaciones, de manera similar a como una firma en las tarjetas de crédito atestigua que el signatario es el legítimo titular. Por su parte, los certificados emitidos a comerciantes equivalen a esas etiquetas mostradas en el escaparate en las que se informa de que aceptan pagos con tarjetas de esta o aquella casa, además de dar fe de su identidad.

Los certificados son emitidos y gestionados por la misma entidad financiera o emisor de tarjetas de la que se recibió la tarjeta de pago. Se necesita un certificado distinto para cada marca diferente de tarjeta de crédito con la que se efectúen las compras (caso del consumidor) o que sea aceptada en el comercio (caso del comerciante). Los certificados SET son emitidos por autoridades de certificación (AC) dentro de la jerarquía de certificación SET (en la que no vamos a entrar). Esta jerarquía asegura la autentificación válida de los participantes. Garantiza además la seguridad de los datos intercambiados entre titulares, comerciantes, bancos y pasarelas de pagos.

Todo esto se refleja dentro del protocolo SET en dos fases previas a la que se ha detallado en el apartado *Cómo funciona SET* y que constituyen las fases de registro tanto del comprador como del vendedor y que también estudiaremos en nuestro modelado.

4.1.6. Ejemplos de fraudes

Quizás la manera más gráfica de comprender algunos de lo mecanismos de SET es ver algunos ejemplos de mejoras respecto a los modelos tradicionales.

- **El comerciante "misssing in the wind"**

  Un "hacker"monta un servidor en Singapur o en algún otro paraíso legal de Internet y se dedica a vender artículos (habitualmente memorias, discos duros,...) a mitad de precio. Por supuesto el servidor utiliza la última versión de SSL con cifrado fuerte. Como la comunicación es segura el comprador incauto adquiere productos facilitando su número de tarjeta de crédito sin pedir más referencias.

  A las dos semanas el servidor desaparece de Internet y, evidentemente, el comerciante también. A partir de ese momento el hacker puede utilizar toda la información que ha adquirido sobre el comprador para su propio uso.

  Nada de esto hubiese pasado si el comprador hubiese exigido un certificado electrónico, expedido por una autoridad de certificación de confianza, que acreditase que ese comerciante era quien decía ser. Es cierto que SSL permite la utilización de certificados, pero por desgracia nadie los exige, ni los comerciantes ni los consumidores en la red. SET, por construcción, obliga a que todos los agentes que intervienen en el proceso de comercio electrónico estén debidamente certificados por una autoridad de confianza.

- **El comprador fantasma**

  El comprador fantasma es una persona con conocimientos de Internet y de como funciona la compra-venta con tarjetas de crédito. Un día hurgando en la basura de su vecino o mirando en su buzón, encuentra una de sus facturas que contiene un número de tarjeta de crédito (un ticket del restaurante o de la gasolinera que normalmente todos tiramos a la papelera sin darle mayor importancia también vale). Con

ese número de tarjeta y dando el nombre de su vecino, el desalmado compra varias docenas de CD en su tienda electrónica favorita y hace que se los envíen a una dirección en la que él sabe que no vive nadie en ese momento. Cuando el cartero llega, como no hay nadie, se deja una nota en el buzón indicando que se puede recoger el paquete en correos, ahora sólo queda hacerse pasar por el vecino para disfrutar de un montón de CD que él ha pagado pero que nunca verá.

Como en el caso anterior, esto no hubiese pasado si el comerciante hubiese pedido un certificado electrónico al comprador, además de su número de tarjeta.

- **Daños colaterales**

  Este es un ejemplo interesante que demuestra que, aunque el comerciante y el comprador sean honrados y pongan toda su buena voluntad, también pueden suceder desastres. Imaginemos esa tienda "*on line*" que tiene miles de compradores que confían en ella. La tienda está basada en varios servidores seguros que utilizan SSL. Los compradores habituales son introducidos en una base de datos que contiene su nombre, dirección y su número de tarjeta de crédito, así cada vez que uno de estos clientes quiere adquirir un producto se le pide solamente las primeras cifras del número de tarjeta para asegurarse de su identidad, evitando así que el número deba ser transmitido de nuevo por la red.

  Desgraciadamente el cortafuegos de la red corporativa del comerciante tiene un agujero de seguridad que permite a un hacker penetrar en su intranet y llegar a la máquina que contiene la base de datos de los clientes. Al día siguiente circula por las news un fichero con miles de números de tarjetas de crédito de honrados compradores.

  En este caso el fallo fundamental es que, dada la arquitectura de SSL, el comerciante debe conocer el número de tarjeta de crédito del cliente para poder cobrar el importe de la venta. SET evita este tipo de problemas haciendo que el comprador cifre su número de tarjeta de crédito con la clave pública de la pasarela de pago. De este modo, el comprador recibe un paquete cifrado que es como un cheque, él no puede ver el número de tarjeta de crédito pero puede guardarlo y enviárselo al banco cuando crea conveniente. El banco, cuando lo reciba su pasarela de pago, podrá descifrarlo y comprobar que el número de tarjeta facilitado corresponde realmente con un número válido. A continuación la cantidad convenida podrá ser transferida a la cuenta del comerciante.

4.2. Especificación del protocolo SET

Como se indicó anteriormente, SET esta compuesto realmente por un conjunto de protocolos, de ellos, los cinco más importantes son:

1. Protocolo de Registro del Comprador (Cardholder Registration)

   Permite a un comprador registrar una tarjeta de crédito con una Autoridad de Certificación para poder realizar transacciones de comercio electrónico seguras.

2. Protocolo de Registro del Vendedor (Merchant Registration)

   Permite a un vendedor registrarse con una Autoridad de Certificación.

3. Protocolo de Petición de Compra (Purchase Request)

   Este protocolo es invocado una vez que el comprador ha terminado de seleccionar los artículos o servicios que ha decidido comprar y se dispone a efectuar el pago mediante una tarjeta de crédito.

4. Protocolo de Autorización de Pago (Payment Authorization)

   Este protocolo sigue, o más concretamente, está combinado con el protocolo de Petición de Compra. Permite a un vendedor verificar los datos del comprador antes de autorizar la transacción.

5. Protocolo de Captura del Pago (Payment Capture)

   Una vez todo esta verificado, queda que el comprador solicite la transferencia de fondos que es de lo que se encarga este último protocolo.

Estos protocolos suelen ser agrupados en dos fase:

- Fase de Registro:
  - Registro del Comprador
  - Registro del Vendedor
- Fase de Compra:
  - Petición de Compra
  - Autorización de Pago
  - Captura del Pago

Cabría pensar, en principio, que la primera fase (Fase de Registro) sólo será necesaria cuando el comprador o el vendedor sienta la necesidad de registrarse para más adelante poder realizar compras seguras en la red utilizando el protocolo SET. Pero desgraciadamente esto no es tan trivial. La autentificación de todas las partes exige rígidas jerarquías de certificados distintos para cada tipo de tarjetas de crédito, esto es, el comprador necesita un certificado distinto para cada tarjeta de crédito con la que pretenda efectuar compras, y el vendedor deberá hacer lo mismo para cada tipo de tarjeta que decida aceptar. Además, es importante destacar que los certificados tienen caducidad por lo que es necesaria su renovación cada cierto tiempo. Así que, aunque evidentemente los protocolos de esta fase no serán tan utilizados como los de la fase de compra, también parece necesario enfrentarnos a su estudio.

Los protocolos que componen la Fase de Compra entran en ejecución cada vez que se realiza una compra por internet utilizando como protocolo de seguridad SET, lo cual, potencialmente, es un número elevado de veces. Si a esto le sumamos el hecho que anteriormente se comentó sobre que el protocolo SET es robusto a costa de ser muy pesado y costoso en ancho de banda requerido y en tiempo de cómputo criptográfico, aparece claramente la necesidad de poder analizar estos protocolos a nivel formal con el objetivo de estudiar distintas configuraciones posibles antes de su implantación para obtener el mayor rendimiento posible con el mínimo número de recursos.

En un protocolo de la envergadura de SET el primer paso para poder realizar este estudio, la especificación de los protocolos, requiere un trabajo muy minucioso. En esta tesis presentamos dos variantes de dicha especificación; una utilizando un único tipo de recurso compartido (Recursos Homogéneos) y otra teniendo dos tipos de recursos compartidos (Heterogéneos). Con la primera versión conseguimos digerir la abundante y detallada documentación proporcionada por los diseñadores de SET, mientras que con la segunda hemos dado un paso más en el intento de que la especificación se aproxime lo mejor posible a los sistemas reales y no se pierdan detalles importantes en la abstracción.

Para la Fase de Registro, después de tener en cuenta diversos aspectos, se llegó a la conclusión de que sólo era necesario utilizar un recurso compartido para su modelado, por lo que no es necesario modificar la especificación para considerar más tipos de recursos compartidos. Pasemos a ver como se consiguieron estas especificaciones en los siguientes apartados.

4.2.1. Fase de Compra

Para que pueda realizarse cualquier transacción económica, una vez realizada la compra (Fase de Compra) es imprescindible que, previamente, los participantes, esto es, el comprador y el vendedor, se hayan registrado con la Autoridad Certificadora correspondiente. Siguiendo este orden temporal parecería más lógico empezar nuestro trabajo de especificación por la fase de registro, sin embargo nosotros empezaremos especificando los protocolos que componen la Fase de Compra. Tomamos esta decisión por que la Fase de Compra requiere un estudio más pormenorizado debido a su mayor complejidad y, principalmente, por el hecho de que los protocolos que la componen se utilizan en cada transacción mientras que los de la Fase de Registro se usan tan sólo cuando un participante quiere registrarse. Recordemos que en la Fase de Compra participan tres protocolos:

- Protocolo de Petición de Compra
- Protocolo de Autorización de Pago
- Protocolo de Captura del Pago

Empezamos estudiando el primero, el protocolo que arranca cuando el comprador ha terminado de seleccionar los artículos/servicios que desea adquirir y se decide a pagarlos: el Protocolo de Petición de Compra del cual podemos ver un esquema general en la Figura 4.1 y que puede ser resumido en los siguientes pasos:

C1.- El comprador realiza una petición de compra al vendedor en la que incluye información sobre su tarjeta de crédito (no el número) y sobre su pedido.

C2.- Cuando el vendedor recibe la petición, le asigna un identificador de transacción único al mensaje.

C3.- El vendedor envía una respuesta al comprador con los certificados, tanto de la pasarela de pago, como el suyo propio.

C4.- El comprador verifica los certificados recibidos del vendedor y de la pasarela de pago.

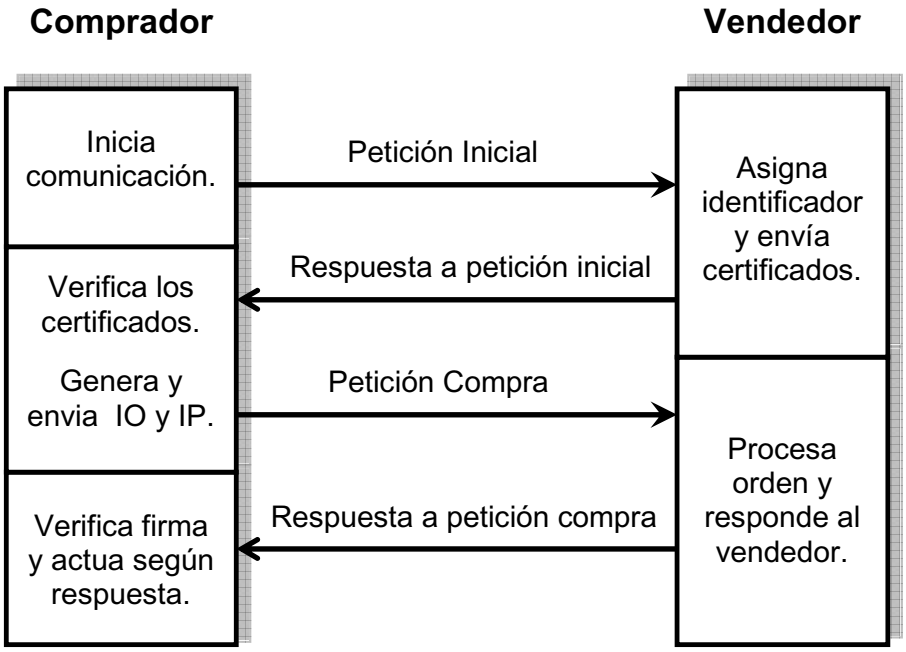


Figura 4.1: Protocolo de Petición de Compra

C5.- El comprador genera la *Información del Pedido* (*Order Information*) y las *Instrucciones de Pago* (*Payment Instructions*). En este punto debemos recordar que la *Información del Pedido* no contiene datos del pedido tales como la descripción de los artículos adquiridos, este tipo de información sólo se intercambia entre el comprador y el vendedor.

C6.- El comprador envía un mensaje con la *Información del Pedido* y las *Instrucciones de Pago* al vendedor.

C7.- El vendedor verifica el certificado y la firma digital del comprador para asegurarse que no ha sido alterado durante la transmisión.

C8.- El vendedor procesa la orden incluyendo la Autorización de Pago.

C9.- El vendedor transmite información al comprador referente a su pedido y si el pago fue rechazado o aprobado. Si el pago fue aceptado, el vendedor deberá enviar los artículos pedidos o realizar los servicios solicitados en el pedido.

C10.- Cuando el comprador recibe el mensaje de respuesta del vendedor, debe verificar su firma y actuar de acuerdo a los contenidos del mensaje.

En el paso *C8*, el vendedor necesita obtener la autorización para la transacción la cual se concede por la Pasarela de Pago. Para realizar esta tarea se utilizan dos protocolos más: el protocolo de Autorización de Pago, que es el que realmente concede o deniega la autorización, y el protocolo de Captura del Pago que será cuando el dinero realmente pase de ser propiedad del comprador a pertenecer al vendedor.

El protocolo de Autorización de Pago se muestra en la Figura 4.2 y puede ser resumido como sigue:

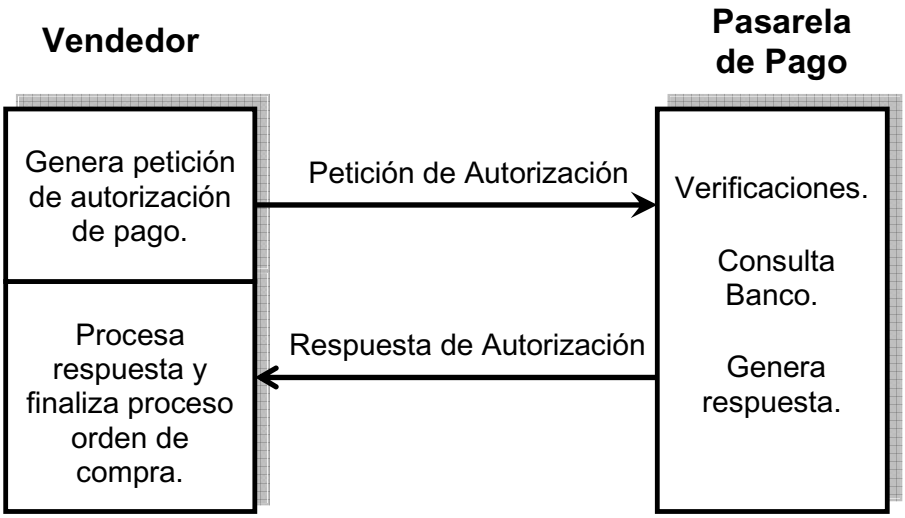


Figura 4.2: Protocolo de Autorización de Pago

A1.- El vendedor genera una petición de autorización que incluye principalmente la cantidad que quiere que se autorice y el identificador de la Información del Pedido.

A2.- El vendedor envía la petición y las instrucciones de pago del comprador a la Pasarela de Pagos.

A3.- Cuando la Pasarela de Pagos recibe la petición de autorización debe verificar varias cosas:

- El certificado del vendedor es correcto y no está caducado.
- La petición fue firmada por el vendedor.
- El certificado del comprador es correcto y no está caducado.

- Las Instrucciones de Pago no han sido modificadas en la transmisión y fueron firmadas por el comprador.
- El identificador de la transacción enviada por el vendedor coincide con el que contiene las Instrucciones de Pago del comprador.

A4.- La Pasarela de Pago envía una petición de autorización al Banco.

A5.- Una vez recibida la respuesta del banco, la Pasarela de Pago la transmite al vendedor.

A6.- El vendedor guarda la autorización (en caso de que haya sido concedida) para usarla posteriormente cuando se la soliciten para poder cobrar el importe acordado (en el Protocolo de Captura del Pago).

A7.- El vendedor completa el proceso requerido en la orden del comprador (paso *C9* del protocolo de Petición de Compra).

Una vez que el vendedor ha atendido el pedido que le solicitó el comprador, llega el momento en el cual el vendedor quiere obtener su pago. Esto se realiza mediante el último de los protocolos: Protocolo de Captura del Pago cuyo esquema mostramos en la Figura 4.3 y que pasamos a resumir:

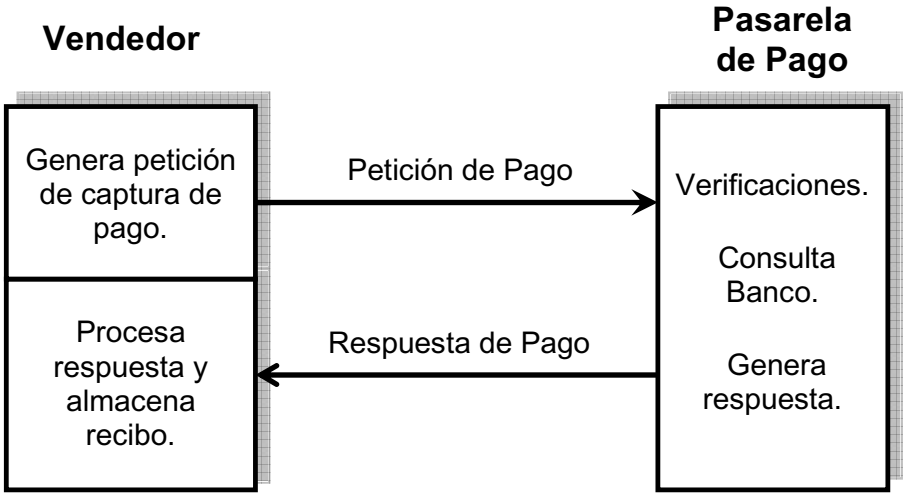


Figura 4.3: Protocolo de Captura del pago

P1.- El vendedor genera una petición de cobro con información sobre la transacción incluyendo la cantidad final de la compra y la envía a la Pasarela de Pagos.

P2.- La Pasarela de Pagos verifica que la petición ha sido firmada por un vendedor autorizado. En caso de ser así, envía las ordenes apropiadas al Banco para que se realice el pago.

P3.- La Pasarela de Pago envía al vendedor la respuesta que le devuelve el Banco.

P4.- El vendedor almacena esta respuesta como recibo que usará para cotejarlo con la cantidad ingresada por el Banco.

4.2.1.1. Recursos Homogéneos

En este punto, somos capaces de saber qué protocolos forman la Fase de Compra y de qué parte en concreto se encarga cada unos de ellos por lo que podemos pasar a preparar la especificación del sistema que modela el comportamiento de estos tres protocolos haciendo uso de nuestro lenguaje **BTC**. Para modelar el sistema que nos ocupa no necesitamos que nuestra álgebra tenga la capacidad de modelar recursos no expropiativos ya que no encontramos ningún recurso de este tipo al modelar el protocolo SET. Por lo tanto, nos decantamos por utilizar la versión de nuestro lenguaje que no cuenta con recursos no expropiativos con el fin de que la especificación final sea lo más legible posible.

Realmente esto no supone ninguna pérdida de generalidad, ya que con la versión definitiva del lenguaje también se puede especificar. De hecho, la versión definitiva es una versión incremental de las anteriores, lo que supone que cualquier cosa que se pueda modelar con una versión anterior podrá ser modelada por la definitiva. Además, en esta primera aproximación, hemos supuesto que contamos con un único tipo de recurso compartido que serán los vendedores reales y que definiremos más adelante. Empecemos viendo cuales son los elementos que toman parte en el sistema y que se están ejecutando en paralelo:

- Un número (m) de clientes (**cardholders**) que han finalizado la elección de los artículos/servicios que desean adquirir.

  Cada uno de ellos debe enviar una petición de inicio de transacción para que el vendedor sepa que ya ha finalizado su compra y que está dispuesto a pagar ($ini\_request_i$). Después el cliente deberá esperar a que le respondan ($ini\_response_i$), momento en el cual deberá verificar los certificados tanto del vendedor como de la Pasarela de Pago (c_verify_i).

Si todas las verificaciones han sido satisfactorias, deberá crear la *Información del Pedido* y las *Instrucciones de Pago* (OI_PI_i) para mandarlas al vendedor ($purchase\_request_i$) después de lo cual deberá volver a esperar respuesta ($purchase\_response_i$) mientras el vendedor procesa la orden y determina si se aprueba o no. Cuando el comprador recibe la respuesta del vendedor sólo le queda verificar que la firma corresponde efectivamente al vendedor (c_verify_i) y hacer las acciones oportunas dependiendo de la respuesta obtenida.

- Un sistema de vendedores virtuales (**merchants**) que proporciona a cada comprador un vendedor virtual.

- Tantos vendedores virtuales $\mathbf{M_i}$ como compradores.

  Existe un vendedor virtual para servir las peticiones de cada nuevo cliente. Este vendedor virtual empieza su ejecución en la sincronización que realiza con el nuevo cliente cuando éste decide comenzar la transacción de pago mediante $ini\_request_i$. Una vez que ha arrancado, lo primero que tiene que hacer es asignar un identificador único a la transacción ($assign\_identifier_i$) y enviarlo al comprador junto con su certificado y el de la Pasarela de Pago ($ini\_response_i$). Después, se queda a la espera de que el comprador genere la *Información del Pedido* y las *Instrucciones de Pago* y se las envíe ($purchase\_request_i$).

  El vendedor realiza todas las verificaciones oportunas (m_verify_i) y genera una petición de autorización ($creat\_autho_i$) que debe mandar a la Pasarela de Pago ($autho\_request_i$) que a su vez debe obtener una autorización del Banco para la transacción en curso y enviarsela al vendedor ($autho\_response_i$). Con esta autorización, el vendedor ya puede completar el proceso de compra con el comprador ($purchase\_response_i$) y más tarde solicitar su cobro a la Pasarela de Pago ($capture\_request_i$).

  En la abstracción del sistema que nosotros estamos realizando, los recursos compartidos (todos del mismo tipo en esta primera aproximación) son los **vendedores reales** de los cuales en un momento dado sólo habrá un número determinado de ellos disponibles en el sistema. Esta restricción en el número de vendedores reales debe ser recogida en nuestro modelo lo cual se refleja en la especificación mediante el conjunto Z. El conjunto Z recoge todas las acciones que para poder ejecutarse necesitan al menos uno de los recursos compartido (esto es, un vendedor real). Estas acciones serán las que acabamos de detallar que debe realizar el vendedor a excepción de las de sincronización.

Así, el conjunto Z estaría compuesto por las siguientes acciones:

$$Z = \{assign\_identifier_i, m\_verify_i, creat\_autho_i\}$$

- Las Pasarelas de Pago (**pay_gateway**) son las encargadas de autorizar las transacciones ($process\_autho_i$) cuando un vendedor se lo solicita ($autho\_request_i$) y de enviarle la respuesta después ($autho\_response_i$). Además, cuando el pedido ha sido servido (o el servicio realizado, si era lo que se adquiría), la Pasarela de Pago recibirá una petición de pago por parte del vendedor ($capture\_request_i$) que quiere cobrar. La Pasarela de Pago se encarga de que este pago se realice ($process\_payment\_capture_i$) y le manda al vendedor la información pertinente ($capture\_response_i$) para que luego éste pueda cotejarla con el recibo del banco.

  En esta abstracción hemos considerado como una única entidad a las Pasarelas de Pago y a los Bancos a pesar de que en el sistema real son diferentes. Esto es así por que estamos trabajando con un álgebra de procesos y podemos aprovecharnos de su modularidad. Esto significa que para poder hacer un estudio del sistema podemos entrar en detalles en una parte de él con el fin de obtener la configuración del sistema que nos proporcione los mejores resultados y más tarde descomponer otra parte de dicho sistema que hayamos tratado como un todo. En esta versión nos centramos en los recursos de tipo vendedor considerando a las pasarelas de pago y a los bancos como una única entidad.

  Las acciones que realiza la Pasarela de Pago que no son de sincronización se ejecutan fuera del sistema que intentamos modelar. Esto es así por que se incluye, por ejemplo, el tiempo que invierte un banco en autorizar una transacción, lo cual, aunque en media se puede hacer una estimación, realmente varía mucho de un banco a otro.

  En cualquier caso, dichos bancos nunca serán recursos modelables en nuestro sistema ya que el diseñador del sistema no tiene ningún control sobre ellos (ni en número ni en características) por lo que cualquier conclusión que obtuviéramos para un mejor funcionamiento no podría ser puesta en funcionamiento. Por lo tanto, el tiempo requerido para ejecutar estas acciones lo hemos considerado desconocido al depender de entidades externas.

Con todo esto, la especificación de la Fase de Compra del protocolo SET considerando que tenemos como único tipo de recurso compartido el vendedor real quedaría como se muestra en la Figura 4.4.

$$Z = \{assign\_identifier_i, m\_verify_i, creat\_autho_i\}$$

$$\{[\![\text{SET\_homo}]\!]\}_{Z,\mathcal{N}} \equiv \{[\![cardholder_1 \parallel \ldots \parallel cardholder_i \parallel \ldots \parallel cardholder_m \parallel merchant \parallel pay\_gateway_1 \parallel \ldots \parallel pay\_gateway_s]\!]\}_{Z,\mathcal{N}}$$

$$cardholder_i \equiv ini\_request_i.ini\_response_i.(c\_verify_i, 1).(OI\_PI_i, 3).purchase\_request_i.purchase\_response_i.(c\_verify_i, 1).cardholder_i$$

$$merchant \equiv M_1 \parallel M_2 \parallel \ldots \parallel M_i \parallel \ldots \parallel M_m$$

$$M_i \equiv ini\_request_i.(assign\_identifier_i, 2).ini\_response_i.purchase\_request_i.(m\_verify_i, 1).(creat\_autho_i, 3).autho\_request_i.autho\_response_i.purchase\_response_i.capture\_request_i.capture\_response_i.M_i$$

$$pay\_gateway_i \equiv autho\_request_i.(process\_autho_i, X).autho\_response_i.capture\_request_i.(process\_payment\_capture_i, Y).capture\_response_i.pay\_gateway_i$$

Figura 4.4: Especificación de la Fase de Compra del Protocolo SET con Recursos Homogéneos

4.2.1.2. Recursos Heterogéneos

En la especificación que acabamos de hacer sólo contamos con un tipo de recurso compartido: los vendedores reales. Dicha especificación es la que hemos utilizado para realizar el análisis de prestaciones en la sección siguiente por que nos pareció que era más fácil analizar los resultados con sólo un tipo de recurso compartido. Pero nuestro lenguaje tiene la capacidad de poder modelar correctamente sistemas que utilicen distintos tipos de recursos compartidos. Por eso en nuestra especificación de la Fase de Compra del protocolo SET damos un paso más con el objetivo de aproximarnos mejor al comportamiento del sistema real y añadimos otro tipo de recurso compartido: las Pasarelas de Pago.

En la primera aproximación al protocolo SET (la realizada con recursos homogéneos) considerábamos que las Pasarelas de Pago y los Bancos constituían una única entidad, por lo que se convertían en entidades externas al

sistema y no formaban parte de los recursos del sistema. Sin embargo, haciendo un análisis más detallado, podemos comprobar que las Pasarelas de Pago si pueden formar parte de nuestro sistema dejando fuera de él solamente a los Bancos. Por lo tanto, incluimos este nuevo tipo de recurso compartido en el modelo. Las Pasarelas de Pago también provocarán retrasos en la vida de los procesos (en el tiempo estimado para finalizar una transacción) dependiendo del grado de utilización mientras que los tiempos necesarios para que los Bancos autoricen o realicen los pagos seguirán considerándose desconocidos.

El proceso que modela la Pasarela de Pago se encarga de proporcionar a cada vendedor una Pasarela de Pago virtual, de modo que cuando un vendedor necesita de un servicio proporcionado por una Pasarela, en principio, tiene la sensación de disponer de una para su uso exclusivo, siendo sólo en los momentos en los que se realizan acciones que requieran para su ejecución dicha pasarela cuando se pueda ver afectado por posibles retrasos ocasionados por la presencia de más vendedores en el sistema. Dichas acciones constituyen el conjunto Z_2 que contiene todas las acciones que no son de sincronización y que necesitan disponer de un recurso de tipo Pasarela de Pago para su ejecución y que definimos como:

$$Z_2 = \{p\_verify_i, p\_verify\_res_i\}$$

Cada Pasarela de Pago virtual está encargada de autorizar (si es posible) una transacción cuando así se le solicita ($autho\_request_i$) e informar al vendedor sobre si ha sido aceptada o denegada ($autho\_response_i$). Cuando se solicita una autorización a una Pasarela de Pago, ésta debe realizar una serie de verificaciones (p_verify_i) que ya se describieron en el paso *A3* de la abstracción del protocolo SET que se hizo en la sección anterior pero que podemos recordar aquí:

- El certificado del vendedor es correcto y no está caducado.
- La petición fue firmada por el vendedor.
- El certificado del comprador es correcto y no está caducado.
- Las Instrucciones de Pago no han sido modificadas en la transmisión y fueron firmadas por el comprador.
- El identificador de la transacción enviada por el vendedor coincide con la que contiene las Instrucciones de Pago del comprador.

Después de efectuar estas verificaciones y comprobar que todo es correcto, la Pasarela de Pago debe de ponerse en contacto con el Banco ($issuer\_autho\_req_i$) para comprobar que éste realmente autoriza dicha transacción (se comprueban cosas como la existencia de fondos, que el crédito máximo no haya sido sobrepasado, etc). El Banco comprueba si puede autorizar la transacción ($autho\_issuer_i$) y genera una respuesta que envía a la Pasarela de Pago ($issuer\_autho\_res_i$).

Más tarde, habitualmente cuando el pedido de compra que se ha realizado se ha servido o los servicios contratados se han realizado o establecido, el vendedor solicita mediante una petición a la Pasarela de Pago ($capture\_request_i$) que se le pague. La Pasarela de Pago verifica que todas las medidas de seguridad se han mantenido (p_verify_i) y envía dicha petición al Banco ($pay\_capture\_req_i$), el cual necesita un tiempo que nosotros tan sólo podemos estimar (depende de cada Banco en particular) para realizar el pago (pay_issuer_i). A continuación, el Banco envía la información a la Pasarela de Pago ($pay\_capture\_res_i$) quien a su vez se la envía al vendedor ($capture\_response_i$) para que tenga constancia de que el pago ha sido realizado.

Añadiendo la nueva especificación de las Pasarelas de Pago a la especificación que habíamos obtenido para el sistema con recursos homogéneos, obtenemos una nueva especificación del sistema que se muestra en la Figura 4.5. Donde recordemos que el conjunto Z ahora está constituido por dos conjuntos Z_1 y Z_2 ya que el sistema cuenta con dos tipos distintos de recursos compartido; los vendedores y las pasarelas de pago.

$$Z_1 = \{assign\_identifier_i, m\_verify_i, creat\_autho_i\}$$
$$Z_2 = \{p\_verify_i, p\_verify\_res_i\}$$

4.2.2. Fase de Registro

Como se ha comentado, previamente a cualquier comunicación entre los participantes en una transacción de comercio electrónico, todas las entidades deben haber obtenido un certificado digital válido a través de la Autoridad de Certificación adecuada, lo cuál se consigue mediante los protocolos de la Fase de Registro:

- Protocolo de Registro del Comprador
- Protocolo de Registro del Vendedor

$$\{[\text{SET\_Hete}]\}_{Z,\ N} \equiv \{[cardholder_1 \parallel \ldots \parallel cardholder_i \parallel \ldots \parallel cardholder_m \parallel merchant \parallel pay\_gateway_1 \parallel \ldots \parallel pay\_gateway_s]\}_{Z,\ N}$$

$$cardholder_i \equiv ini\_request_i.ini\_response_i.(c\_verify_i, 1).(OI\_PI_i, 3).purchase\_request_i.purchase\_response_i.(c\_verify_i, 1).cardholder_i$$

$$merchant \equiv M_1 \parallel M_2 \parallel \ldots \parallel M_i \parallel \ldots \parallel M_m$$

$$M_i \equiv ini\_request_i.(assign\_identifier_i, 2).ini\_response_i.purchase\_request_i.(m\_verify_i, 1).(creat\_autho_i, 3).autho\_request_i.autho\_response_i.purchase\_response_i.capture\_request_i.capture\_response_i.M_i$$

$$pay\_gateway_i \equiv P_1 \parallel P_2 \parallel \ldots \parallel P_i \parallel \ldots \parallel P_m$$

$$P_i \equiv autho\_request_i.(p\_verify_i, 5).issuer\_autho\_req_i.(autho\_issuer_i, X).issuer\_autho\_res_i.autho\_response_i.capture\_request_i.(p\_verify\_res_i, 1).pay\_capture\_req_i.(pay\_issuer_i, Y).pay\_capture\_res_i.capture\_response_i.pay\_gateway_i$$

Figura 4.5: Especificación de la Fase de Compra del Protocolo SET con Recursos Heterogéneos

4.2.2.1. Registro del Comprador

En la Figura 4.6 se muestra una perspectiva general del proceso llevado a cabo por el protocolo de Registro del Comprador, el cuál se compone de tres intercambios de mensajes entre el Comprador y la Autoridad de Certificación. En el primer intercambio, el comprador solicita registrarse y la Autoridad de Certificación le proporciona la clave para certificarse con la que podrá proteger el número de cuenta bancario en el formulario de registro.

En el segundo intercambio, el comprador solicita un formulario para registrarse. Para ello proporciona, de manera codificada y gracias a la clave que le ha sido proporcionada, su número de tarjeta de crédito (PAN por Primary Account Number) con la que la Autoridad de Certificación será capaz de identificar el banco que la mantiene y le mandará el formulario adecuado.

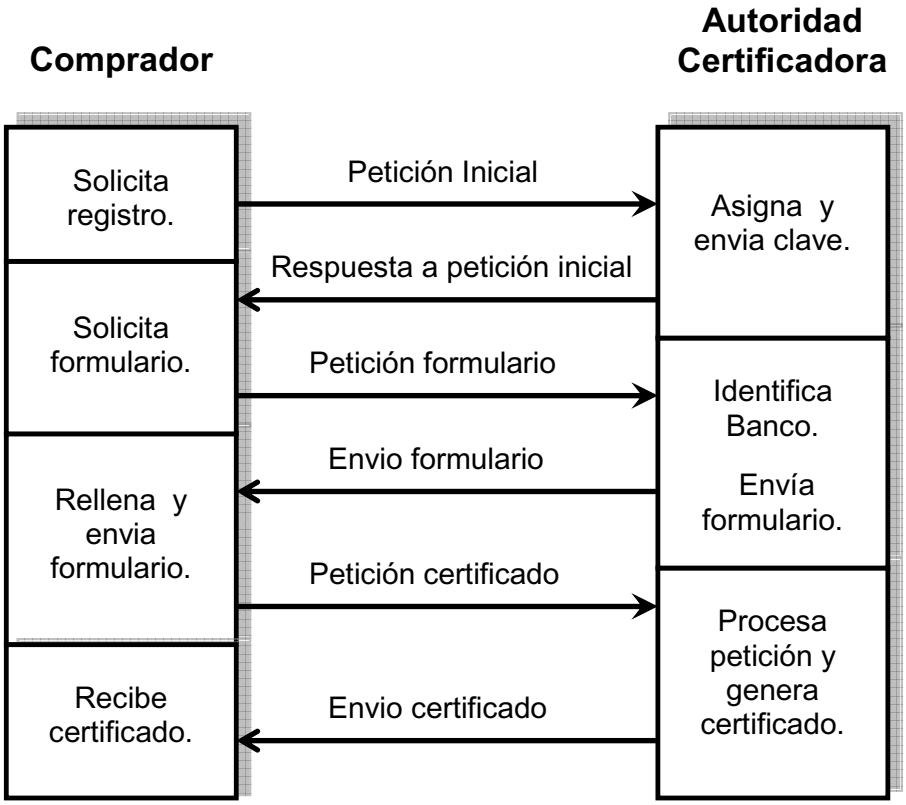


Figura 4.6: Protocolo de Registro del Comprador

En el tercer paso, el comprador devuelve completado el formulario de registro y recibe un certificado que contiene la clave, la firma digital y todo lo necesario para poder autentificarse en el futuro en cualquier transacción de comercio electrónico que use como protocolo de seguridad SET.

En este protocolo podemos ver que sólo existe un tipo de recurso compartido: las Autoridad de Certificación reales disponibles. Y el conjunto formado por las acciones que necesitan un recurso de este tipo para su ejecución vendrá dado por:

$$Z_1 = \{gen\_key_i, select\_bank_i, gen\_certificate_i\}$$

que no pasamos a explicar por que los nombres que hemos dado a dichas acciones son suficientemente significativos. La especificación de este protocolo se puede ver en la Figura 4.7.

$$\{[\text{Reg\_cardholder}]\}_{Z,\mathcal{N}} \equiv \{[cardholder_1 \parallel \ldots \parallel cardholder_i \parallel \ldots \parallel cardholder_m \parallel cert\_autho]\}_{\{\{gen\_key_i, select\_bank_i, gen\_certificate_i\}\},\mathcal{N}}$$

$$cardholder_i \equiv ini\_key_i.res\_key_i.(PAN\_message_i, 3).\ ini\_form_i.res\_form_i.(fill\_form_i, 20).\ ini\_certificate_i.res\_certificate_i.cardholder_i$$

$$cert\_autho \equiv C_1 \parallel C_2 \parallel \ldots \parallel C_i \parallel \ldots \parallel C_m$$

$$C_i \equiv ini\_key_i.(gen\_key_i, 2).res\_key_i.ini\_form_i.(select\_bank_i, 5).\ res\_form_i.ini\_certificate_i.(gen\_certificate_i, 10).\ res\_certificate_i$$

Figura 4.7: Especificación del Protocolo de Registro del Comprador

4.2.2.2. Registro del Vendedor

El protocolo de Registro del Vendedor es similar al del comprador pero más sencillo ya que sólo consta de dos intercambios de mensajes en los que además no es necesario transmitir ningún número de tarjeta de crédito. Los cinco principales pasos de este protocolo quedan reflejados en la Figura 4.8.

El proceso de registro del comerciante arranca cuando éste solicita un formulario de registro a la Autoridad de Certificación, la cual se lo envía para que lo rellene y lo devuelva. Una vez que la entidad certificadora recibe el formulario le enviará al vendedor el certificado.

En este protocolo, igual que en el de Registro del Comprador, sólo existe un tipo de recurso compartido que serán las Autoridad de Certificación disponibles. Aquí, el conjunto formado por las acciones que necesitan un recurso de este tipo para su ejecución vendrá dado por:

$$Z_1 = \{select\_bank_i, gen\_certificate_i\}$$

y la especificación de este protocolo se encuentra en la Figura 4.9.

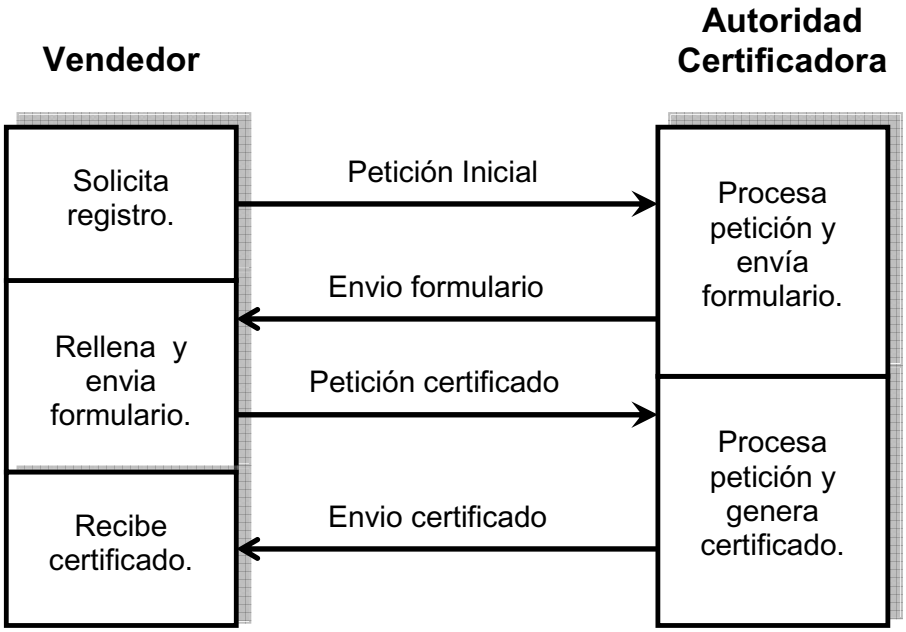


Figura 4.8: Protocolo de Registro del Vendedor

$$\{[\text{Reg\_merchant}]\}_{Z,\mathcal{N}} \equiv \{[merchant_1 \parallel \ldots \parallel merchant_i \parallel \ldots \parallel merchant_m \parallel cert\_autho]\}_{\{\{select\_bank_i, gen\_certificate_i\}\},\mathcal{N}}$$

$$merchant_i \equiv ini\_form_i.res\_form_i.(fill\_form_i, 20).ini\_certificate_i.res\_certificate_i.merchant_i$$

$$cert\_autho \equiv C_1 \parallel C_2 \parallel \ldots \parallel C_i \parallel \ldots \parallel C_m$$

$$C_i \equiv ini\_form_i.(select\_bank_i, 5).res\_form_i.ini\_certificate_i.(gen\_certificate_i, 10).res\_certificate_i$$

Figura 4.9: Especificación del Protocolo de Registro del Vendedor

4.3. Análisis de prestaciones en SET

Con el modelo formal que hemos definido somos capaces de capturar las características temporales del sistema. Ahora vamos a utilizar el algoritmo de análisis de prestaciones definido en el capítulo anterior para estimar el tiempo que se necesita para que el sistema evolucione de un estado a otro, y más concretamente el tiempo necesario para alcanzar el estado final desde el inicial.

Para hacer este estudio y con el fin de que los resultados obtenidos fueran fácilmente comprensibles, hemos tomado la especificación de la Fase de Compra del protocolo SET con recursos homogéneos. Así, partiendo de esta especificación del sistema y gracias a la semántica operacional, somos capaces de crear el grafo de transición de estados que nos ayudará en el estudio de prestaciones.

Representar el grafo de transición de estados de forma que se pueda apreciar su información claramente no es una tarea fácil debido al gran número de estados que contiene. Hemos supuesto que tenemos en el sistema tan solo cinco compradores y que el número de vendedores reales disponibles es de tres. Con estas premisas, hemos construido el grafo de transición de estados correspondiente que consiste en casi 520 nodos. No es un grafo excesivamente grande pero realmente presenta ciertas dificultades para mostrarse en una figura de modo que quede legible. Este grafo podría ser reducido eliminando los estados equivalentes, pero aún así el grafo resultante contiene 340 nodos (estados) que todavía presenta dificultades para ser representado. Por eso hemos elegido tres nodos de dicho grafo y los hemos representado en la Figura 4.10 con el fin de poder mostrar parte de este grafo y, sobre todo, su forma de evolucionar. Además, para mayor claridad en la figura hemos omitido el valor de los nodos que se mostrará más adelante.

En cada uno de estos nodos, también por claridad, sólo se muestra la siguiente acción que cada proceso puede ejecutar (lo cual no es una pérdida de información si no tan solo una simplificación a la hora de representarlo, esto es, el resto de acciones se da por supuesto que también están). También en los nodos, hemos representado en **negrita** las acciones que pueden ser ejecutadas, mientras que las que necesitan algún recurso para su ejecución las hemos marcado con un asterisco.

Hemos elegido precisamente esta parte del grafo por que podemos ver en él algunas cosas bastantes interesantes. Suponemos que el sistema se encuentra en el estado que hemos llamado N_0 el cual consta de las siguientes acciones ejecutándose en paralelo:

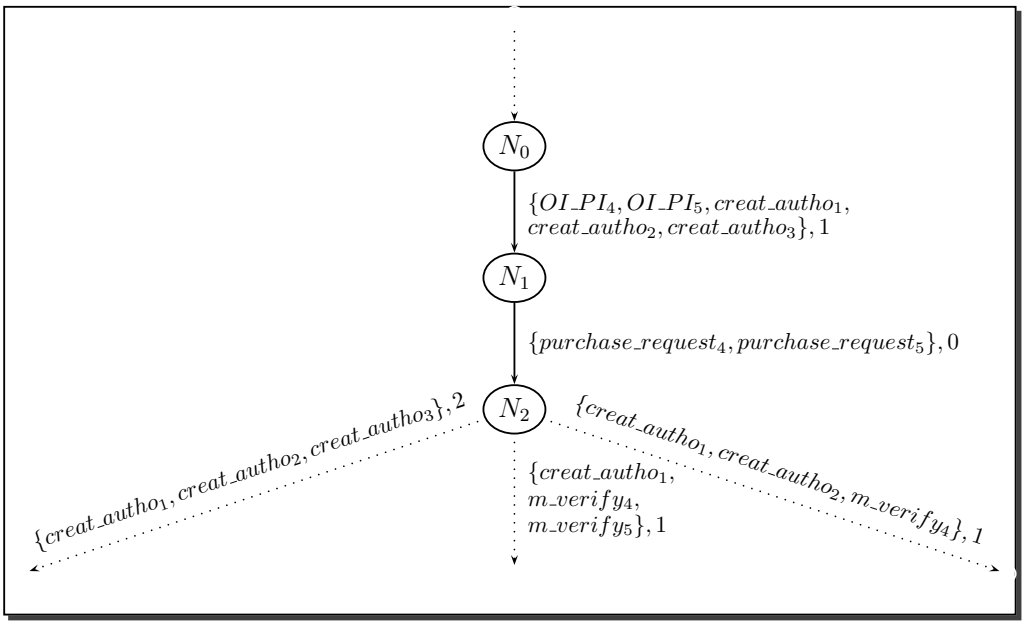


Figura 4.10: Zoom del Grafo de Transiciones de la Fase de Compra del Protocolo SET con cinco procesos Compradores y Vendedores.

	$\mathbf{N_0}$(acciones en paralelo)	
$\mathbf{cardholders_i}$	$\mathbf{M_i}$(**Merchants**)	$\mathbf{pay\_gateway_i}$
$purchase\_response_1$	$*(\mathbf{creat\_autho_1}, \mathbf{3})*$	$autho\_request_1$
$purchase\_response_2$	$*(\mathbf{creat\_autho_2}, \mathbf{3})*$	$autho\_request_2$
$purchase\_response_3$	$*(\mathbf{creat\_autho_3}, \mathbf{3})*$	$autho\_request_3$
$(\mathbf{OI\_PI_4}, \mathbf{1})$	$purchase\_request_4$	$autho\_request_4$
$(\mathbf{OI\_PI_5}, \mathbf{1})$	$purchase\_request_5$	$autho\_request_5$

Veamos realmente que significa. En este momento, en el sistema hay 15 procesos ejecutándose; cinco procesos *cardholder* (compradores), cinco M (vendedores virtuales) y cinco *pay_gateway* (pasarelas de pago). La próxima acción que tiene que ejecutar, por ejemplo, el proceso $cardholder_1$ es $purchase\_response_1$ que se está ejecutando en paralelo con el resto de las acciones representadas arriba, las cuales hemos representado en columnas sólo para que se vea más claramente.

Las acciones que están preparadas para ejecutarse están, como se ha dicho anteriormente, marcadas en negrita. Siguiendo las reglas de la semántica operacional se decide cuales de esas acciones realmente se podrán ejecutar. Dichas acciones serán:

- Todas las acciones que no pertenezcan al conjunto Z, esto es, todas las acciones que no necesiten ningún recurso compartido para su ejecución. En este nodo $(OI\_PI_4, 1)$ y $(OI\_PI_5, 1)$.
- Todas las acciones de sincronización que puedan sincronizar. En este nodo no hay ninguna que pueda hacerlo. Las acciones de sincronización están representadas en *cursiva* y como vemos no hay dos acciones de sincronización iguales para poder sincronizar.
- Hasta un máximo de $\mathcal{N}$ acciones que necesiten de un recurso compartido. En este sistema contamos con 3 vendedores reales (el recurso compartido) por lo que $\mathcal{N}$ vale 3 y éste será el número máximo de acciones de este tipo que podrán ejecutarse simultáneamente. En el nodo N_0 tenemos sólo tres acciones de este tipo preparadas por lo que podrán ejecutarse las 3: $(creat\_autho_1, 3)$, $(creat\_autho_2, 3)$ y $(creat\_autho_3, 3)$

Después de esta evolución que usará una unidad de tiempo (es la duración de la acción más corta a ejecutar en este paso) el sistema se encuentra en el estado N_1:

		$\mathbf{N_1}$(acciones en paralelo)	
$\mathbf{cardholders_i}$		$\mathbf{M_i}$(**Merchants**)	$\mathbf{pay\_gateway_i}$
$purchase\_response_1$		$*(\mathbf{creat\_autho_1}, \mathbf{2})*$	$autho\_request_1$
$purchase\_response_2$		$*(\mathbf{creat\_autho_2}, \mathbf{2})*$	$autho\_request_2$
$purchase\_response_3$		$*(\mathbf{creat\_autho_3}, \mathbf{2})*$	$autho\_request_3$
$\mathbf{purchase\_request_4}$	$\Longleftrightarrow$	$\mathbf{purchase\_request_4}$	$autho\_request_4$
$\mathbf{purchase\_request_5}$	$\Longleftrightarrow$	$\mathbf{purchase\_request_5}$	$autho\_request_5$

Aquí tenemos acciones de sincronización que pueden ejecutarse (marcadas con $\Longleftrightarrow$), como su duración es de cero unidades de tiempo son las primeras que elegimos para su ejecución, haciendo que el sistema evolucione al estado N_2:

	$\mathbf{N_2}$(acciones en paralelo)	
$\mathbf{cardholders_i}$	$\mathbf{M_i}$(**Merchants**)	$\mathbf{pay\_gateway_i}$
$purchase\_response_1$	$*(\mathbf{creat\_autho_1}, \mathbf{2})*$	$autho\_request_1$
$purchase\_response_2$	$*(\mathbf{creat\_autho_2}, \mathbf{2})*$	$autho\_request_2$
$purchase\_response_3$	$*(\mathbf{creat\_autho_3}, \mathbf{2})*$	$autho\_request_3$
$purchase\_response_4$	$*(\mathbf{m\_verify_4}, \mathbf{1})*$	$autho\_request_4$
$purchase\_response_5$	$*(\mathbf{m\_verify_5}, \mathbf{1})*$	$autho\_request_5$

En este estado, cinco acciones que necesitan un recurso compartido están preparadas para ejecutarse, más que recursos compartidos (sólo tenemos tres). Como consecuencia el sistema tiene varias evoluciones posibles dependiendo de qué tres acciones de esas cinco seleccione para ejecutar en el siguiente paso, así el grafo deberá considerar todas las posibles evoluciones para más tarde poder ser estudiadas.

Nosotros hemos generado estos grafos de transición de estados para distintos valores tanto en el número de compradores como en el número de recursos compartidos (vendedores reales) con el fin de poder evaluar y comparar los requerimientos temporales necesarios para evolucionar desde el estado inicial al final. Los resultados obtenidos se ajustan perfectamente a los esperados en el sistema real; manteniendo fijo el número de compradores en el sistema, si se va incrementando el número de vendedores reales en el sistema, el tiempo necesario para alcanzar el estado final se decrementa rápidamente al principio, luego más lentamente y finalmente se queda estable o se decrementa mínimamente con lo que la mejoría es casi inapreciable.

Estos resultado se muestran en la Tabla 4.1 donde a cada valor obtenido para el tiempo necesario para finalizar las transacciones iniciadas por los compradores habría que sumarle el tiempo invertido por las acciones que no son de sincronización (ya que las de sincronización tienen tiempo 0) realizadas por las pasarelas de pago, esto es X+Y. Pero como este valor permanece fijo en todos los casos se ha optado por no ponerlo y mantener la claridad de la tabla.

Estos resultados resultan de gran ayuda a la hora de tomar decisiones cuando se está diseñando el sistema. Por ejemplos, si estimamos que el número de compradores en el sistema es seis, observamos que el tiempo necesario para atenderlos, esto es el tiempo necesario para alcanzar el estado final, es el mismo si contamos con cuatro o cinco vendedores reales. Con esta infor-

Compradores	Vendedores	Mínimo Tiempo
1	1	10
2	1	14
2	2	10
3	1	18
3	2	12
3	3	10
4	1	24
4	2	14
4	3	12
4	4	10
5	1	30
5	2	16
5	3	13
5	4	12
5	5	10
6	1	36
6	2	18
6	3	14
6	4	12
6	5	12
6	6	10
7	1	42
7	2	21
7	3	15
7	4	13
7	5	12
7	6	12
7	7	10
8	1	48
8	2	24
8	3	16
8	4	14
8	5	13
8	6	12
8	7	12
8	8	10
9	1	54
9	2	27
9	3	18
9	4	15
9	5	14
9	6	12
9	7	12
9	8	12
9	9	10
10	1	60
10	2	30
10	3	20
10	4	16
10	5	14
10	6	13
10	7	12
10	8	12
10	9	12
10	10	10

Cuadro 4.1: Tiempo necesario para alcanzar el estado final con distinto valores en el número de compradores y vendedores

mación parece claro que no tendría mucho sentido implementar el sistema con ese quinto vendedor ya que no obtenemos ninguna mejoría con él. Evidentemente, éste sería sólo uno de los parámetros que ayudarían a la toma de las decisiones finales, pero parece evidente que arroja una información

a tener en cuenta. En otros casos no es tan clara la decisión a tomar, por ejemplo, si estimamos que el número medio de compradores en el sistema es diez, podemos comprobar, por los resultados recogidos en la tabla, que existe una ligera mejoría si el sistema cuenta con seis vendedores en lugar de con cinco. En este caso la mejoría no es tan grande como en el caso anterior por lo que se requerirá un análisis posterior para estimar si esta ganancia en tiempo compensa con el coste de aumentar el número de vendedores.

Con el objeto de hacer esta información más legible hemos pensado que sería una buena idea pasarla a un gráfico, así podemos ver los resultados obtenidos en la Tabla 4.1 representados gráficamente en la Figura 4.11. En este gráfico hemos representado con una línea las distintas posibilidades en la estimación del número de compradores simultáneos en el sistema y podemos ver claramente que los resultados obtenidos siguen el mismo patrón que los obtenidos experimentalmente, esto es, cuantos más vendedores reales tenemos (más recursos compartidos) menos tiempo se necesita para servir las peticiones de los compradores hasta que llega un punto en el que aunque se aumente el número de vendedores no se obtiene ningún beneficio.

El gráfico situado en la parte inferior de la Figura 4.11 muestra una ampliación que nos permite estudiar un caso en particular; supongamos que en el sistema no podemos tener más de cuatro vendedores y que los requerimientos temporales que se imponen al sistema es que no se puede tardar más de 14 unidades de tiempo en alcanzar el estado final. Hemos delimitado con una línea horizontal la zona que cumple estos requerimientos y observamos que contamos con varias posibilidades. Podemos atender hasta 8 compradores satisfactoriamente si disponemos de cuatro vendedores, pero también es posible contar con sólo tres vendedores y ser capaces de atender a seis compradores simultáneamente. Sabemos que para la toma de decisiones no basta con tener estos datos en consideración y que hace falta tener en cuenta otros parámetros como el coste de cada nuevo recurso (vendedor), las posibilidades de que el número de compradores aumente, etc. Pero en cualquier caso, repetimos, esta información nos proporciona un valioso dato a tener en cuenta cuando se va a determinar el número de recursos en un sistema.

Con este mismo modelo del protocolo SET hemos sido capaces de estudiar otro aspecto del sistema. Dado un número fijo de vendedores en el sistema, vamos a evaluar el rendimiento del sistema con diferentes valores en el número de compradores. Los resultados obtenidos con esta simulación nos permitirán determinar el número máximo de compradores que podemos admitir en el sistema manteniendo unos requerimientos temporales. Calculamos nuevamente los tiempos resultantes mediante nuestro grafo de estados y

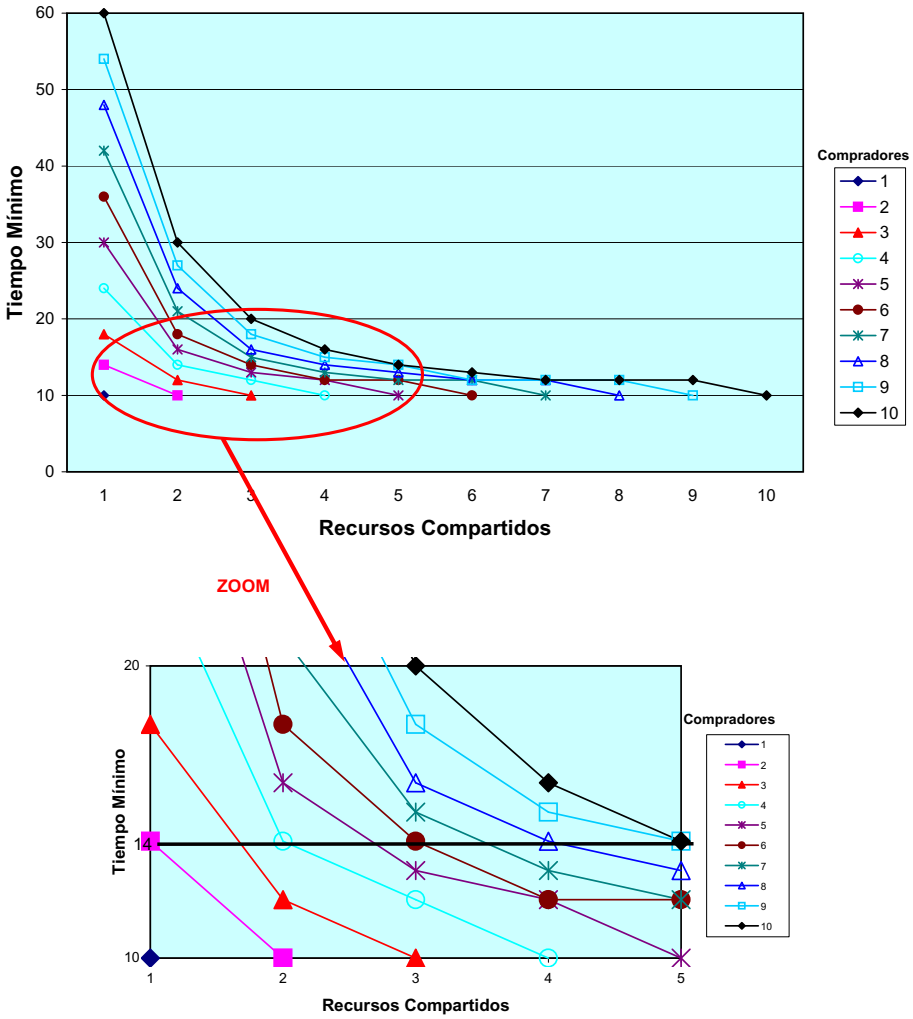


Figura 4.11: Resultados obtenidos a partir de la Tabla 4.1.

el algoritmo de evaluación de prestaciones y obtenemos los resultados mostrados en la Tabla 4.2 de la que podemos sacar conclusiones suficientes para determinar el número de compradores que el sistema es capaz de servir dependiendo de los requerimientos temporales que tenga. Por ejemplo, si el sistema que tenemos que implementar tiene como requerimientos temporales que no puede tardar más de 20 unidades de tiempo en alcanzar su estado final, miramos la Tabla 4.2 y observamos que este sistema no debería permitir la entrada a más de 16 compradores simultáneamente si quiere mantener esos requerimientos temporales.

Número Compradores	**Mínimo Tiempo**
1	10
2	10
3	10
4	10
5	10
6	12
7	12
8	13
9	14
10	14

Número Compradores	**Mínimo Tiempo**
11	15
12	16
13	17
14	18
15	18
16	20
17	21
18	22
19	23
20	24

Número Compradores	**Mínimo Tiempo**
25	30
30	36
35	42
40	48
45	54
50	60

Cuadro 4.2: Evaluación de Prestaciones en un sistema con 5 Vendedores

Capítulo 5

Aplicación: Sistemas de Fabricación Flexibles

5.1. Qué es un Sistema de Fabricación Flexible

A mediados de los sesenta, la competencia en los mercados se intensificó de manera significativa. Entre 1960 y 1970 el *coste* fue la primera preocupación, pero más tarde fue la *calidad* lo que adquirió más protagonismo y prioridad. El mercado cada vez era más complejo y la *velocidad de entrega* se convirtió pronto en algo prioritario para los clientes. Apareció una nueva estrategia:*Customizability*, esto es, diversificación del producto para adaptarse a las necesidades especificas del cliente. Las compañías, por lo tanto, deben adaptarse al entorno en el que operan y ser más *flexibles* en sus operaciones ya que, ahora, los productos tienen un ciclo de vida relativamente corto, debido a las constantes incorporaciones de nuevas y sofisticadas tecnologías. La flexibilidad del producto y de los procesos de fabricación juega un papel importante en la competitividad de las empresas.

Con esto podemos ver que la aparición e innovación de los Sistemas de Fabricación Flexibles (FMS por *Flexible Manufacturing Systems*) tuvo que ver principalmente con el esfuerzo realizado para obtener ventajas en la fiera competencia aparecida en el mercado.

Principalmente, un FMS es una tecnología de fabricación, pero también es una filosofía. Filosóficamente, un FMS incorpora una nueva visión del sistema de fabricación. La palabra clave para la fabricación de hoy en día es *agilidad*. Un fabricante ágil es el más rápido del mercado, opera con los

costes totales más bajos y tiene la gran habilidad de complacer a sus clientes. Los FMS son simplemente un medio por el que los fabricantes pueden llegar a conseguir esta agilidad.

Un sistema de fabricación flexible (FMS) consiste en un grupo de máquinas controladas por computadoras y sistemas automáticos de manejo, carga/descarga de material, y de operación directa sobre el material; todo ello, controlado por un computador supervisor. Los elementos de este sistema son muy flexibles y versátiles, lo que permite una fabricación de distintos tipos de productos simultáneamente.

5.1.1. Concepto de Flexibilidad. Diferentes visiones

Para nosotros, en el campo que nos ocupa, la definición que más se ajustaría a la palabra flexibilidad sería "producir productos de alta calidad, a un precio razonable, que se adapten a la demanda actual de los clientes y que puedan ser rápidamente servidos". Pero para ir situando nuestro trabajo en el extenso campo de los FMS, veamos primero en la Tabla 5.1 diferentes aproximaciones para el concepto de "flexibilidad" y sus significados.

	Significado de Flexibilidad
Fabricación	* Capacidad de producir diferentes piezas sin demasiadas reestructuraciones. * Medida de cómo de rápido las compañías adaptan los procesos de fabricación de antiguos productos a los nuevos. * Habilidad de cambiar un programa de producción para modificar una pieza o manejar múltiples piezas.
Operacional	Habilidad de producir de una manera eficiente productos. altamente personalizados.
Cliente	Capacidad de adaptación a diferentes velocidades de entrega.
Estratégico	Habilidad de una compañía para ofrecer una amplia variedad de productos a sus clientes.
Capacidad	Capacidad de incrementar o decrementar rápidamente el nivel de producción y de poder pasar de un producto a otro.

Cuadro 5.1: Significados de Flexibilidad

Podíamos resumir que al tratar con FMS, el adjetivo "flexible" parece centrarse en la habilidad del sistema para responder eficientemente a los cambios. Pero esta definición todavía parece poco precisa en cuanto a que no termina de centrarnos en donde vamos a realizar nuestro esfuerzo. Veamos una clasificación bastante clara de los distintos niveles de flexibilidad que existen en la fabricación:

A.- Flexibilidad básica

- Flexibilidad de la máquina (Machine flexibility): facilidad con la que una máquina puede procesar diferentes operaciones.
- Flexibilidad en el manejo de materiales (Material handing flexibility): medida de la facilidad con la que diferentes tipos de piezas pueden ser transportadas y colocadas adecuadamente en las distintas partes de las máquinas que constituyen un sistema.
- Flexibilidad en las operaciones (Operation flexibility): medida de la facilidad con la que secuencias de operaciones alternativas se pueden usar para procesar un tipo de pieza.

B.- Flexibilidad del sistema

- Flexibilidad en el volumen (Volume flexibility): medida de la capacidad del sistema para trabajar adecuándose a diferentes volumenes de cada tipo de pieza.
- Flexibilidad de expansión (Expansion flexibility): habilidad de construir un sistema y expandirlo incrementalmente.
- Flexibilidad en el encaminamiento (Routing flexibility): medida de los caminos alternativos que una pieza puede realmente seguir a través de un sistema para un plan de procesamiento concreto.
- Flexibilidad del proceso (Process flexibility): medida del volumen del conjunto de tipos de piezas que un sistema puede producir sin necesidad de reinicializarse.
- Flexibilidad en la producción (Product flexibility): volumen del conjunto de tipos de piezas que pueden ser producidas en un sistema con la menor reorganización.

C.- Flexibilidad total

- Flexibilidad del programa (Program flexibility): capacidad de un sistema para funcionar durante periodos de tiempo razonablemente largos sin intervención externa.
- Flexibilidad en la producción (Production flexibility): volumen del conjunto de tipos de piezas que el sistema puede producir sin tener que invertir mucho capital en equipos.
- Flexibilidad de mercado (Market flexibility): capacidad de un sistema de adaptarse eficientemente a los cambios en las condiciones del mercado.

Así, recogiendo esos distintos niveles de flexibilidad, podemos decir que un FMS es flexible debido a que es capaz de realizar distintas piezas o productos diferentes de forma simultanea en sus estaciones de trabajo. Dependiendo de la demanda es capaz de variar entre diferentes tipos de productos y la tasa de producción de los mismos.

Para que esta idea no quede sólo en una definición se ha diseñado un cuestionario que deben pasar los FMS para garantizar su flexibilidad. Este cuestionario consta de las siguientes preguntas:

- Test de variedad de productos. ¿Puede el sistema procesar diferentes tipos de piezas en un modo de producción que no sea por lotes?
- Test de cambio de programación o producción. ¿Puede el sistema realizar cambios en la producción programada y cambios en cualquier parte del producto o de las cantidades fabricadas?
- Test de recuperación de errores. ¿Se puede recuperar el sistema satisfactoriamente de roturas o errores de funcionamiento sin que esto conlleve la interrupción completa de la producción?
- Test sobre la posibilidad de ampliar la gama de fabricación de piezas o partes producidas. ¿Se pueden añadir nuevas piezas diseñadas al sistema de fabricación de manera relativamente fácil?

Si la respuesta, al menos a las tres primeras preguntas, es positiva podemos afirmar que estamos trabajando un Sistema de Fabricación que es flexible.

5.1.2. Beneficios/Costes de los FMS

Ya sabemos lo que es un Sistema de Fabricación, tenemos una idea de lo que ha significado la adaptación del concepto de Flexibilidad a ellos, pero nos queda alguna cuestión inevitable más: esta flexibilidad tendrá un coste (tanto temporal como económico) y, sin entrar en muchos detalles, parece evidente que no será pequeño. Entonces nos preguntamos si realmente los beneficios obtenidos con las FMS compensan suficientemente como para adoptar esta nueva metodología. Veamos a continuación cuales son estos beneficios frente a los costes que conlleva y descubriremos que es una adaptación necesaria tanto más cuando nuestro trabajo con **BTC** precisamente va a centrarse principalmente en reducir, en la medida de lo posible, esos costes. Por lo tanto, empecemos viendo los principales beneficios de los FMS:

- Incremento del uso de máquinas: Utilizando la tecnología de los FMS, el uso de cada una de las máquinas de nuestro sistema puede ascender al 80-90 %, debido a diferentes razones como:
  - Posibilidad de tener un régimen de operación de 24 horas al día.
  - Cambio de herramientas automático en las máquinas.
  - El cambio de palés automático en las estaciones es más rápido y efectivo.
  - La posibilidad de tener colas de piezas en las estaciones elimina la probabilidad de inhibición en las máquinas.
  - Una programación dinámica de la producción que tiene en cuenta irregularidades en el régimen de operación normal.
- Necesidad de menos máquinas: Al incrementar el uso de cada máquina, se necesita un menor número de ellas para la misma tasa de producción.
- Reducción de la superficie de la fábrica: Generalmente los FMS requieren un 40-50 % menos de superficie que un sistema *job shop* [1] para la misma capacidad.
- Muy receptivo a los cambios: Un FMS mejora la capacidad de respuesta a cambios en el diseño, introducción de nuevos tipos de piezas, cambios

[1] Configuración de Taller (*Job-shop*). Consiste en una fabricación no en serie, de lotes pequeños, para pedidos únicos o de pequeñas cantidades. Por lo regular implica productos adaptados, diseñados a la medida del cliente y de naturaleza muy poco repetitiva. Como se fabrican productos muy diferentes, los recursos son flexibles y versátiles.

en la cadena de producción, respuesta rápida ante averías en máquinas y herramientas. Los ajustes se pueden hacer en la cadena de producción de un día para otro para cubrir ciertas demandas y peticiones del cliente.

- Reducción de la necesidad de inventarios: Debido a que los diferentes tipos de piezas se procesan juntas, y no en lotes separados, el WIP (work-in-process) es menor que en una producción por lotes. La estimación de esta reducción es de un 60-80 %.

- Tiempos de fabricación reducidos: Estrechamente relacionado con el WIP tenemos el tiempo empleado en el proceso por cada una de las piezas. Esto se traduce en una mayor rapidez en la entrega al cliente.

- Menor necesidad de mano de obra y mayor productividad: Mayor tasa de producción y menor dependencia de mano de obra significa una mayor productividad por hora de trabajo con los FMS respecto a otros métodos convencionales.

- Oportunidad para la producción desatendida: El alto nivel de automatización en un FMS le permite funcionar durante periodos extensos de tiempo sin atención humana. Normalmente estos periodos de tiempo coinciden con la noche, o los fines de semana, cuando no haya operarios en la planta.

Como vemos los beneficios son muchos pero es necesario enfrentarlos con sus costes para saber si realmente vale la pena adoptar esta nueva metodología. Los costes que supone la implantación y utilización de los FMS son:

- Existe una limitación en la capacidad de adaptación por parte de las máquinas a la fabricación de nuevos productos (evidentemente las máquinas y herramientas tienen una funciones predeterminadas y hay que ceñirse a ellas).

- Es necesaria una actividad considerable en la preinstalación (intentar diseñar la configuración que mejor se adapte a las necesidades actuales y obtenga mejores rendimientos).

- Económicamente la inversión en nuevas tecnologías no suele ser pequeña.

- Suelen surgir problemas tecnológicos debido al acto grado de exactitud requerida en el posicionamiento de piezas (componentes) y los tiempos necesarios para su procesamiento.

En minimizar algunos de estos *costes* es en lo que nosotros vamos a trabajar. En concreto creemos que podemos minimizar considerablemente el tiempo necesario para realizar/modificar el diseño del sistema para que se adapte a nuevas demandas. Presentamos especificaciones de los sistemas en las que los tiempos de procesamiento quedan claramente definidos y lo más importante de todo, permitimos hacer un estudio de prestaciones antes de tener el sistema implantado. Este hecho es de vital importancia ya que al poder hacer un estudio comparativo de distintas configuraciones de un sistema, sólo se implantará la opción que obtenga mejores resultados. Parece evidente que poder realizar este estudio previo repercute de una manera directa en el coste de la inversión, tema que sin duda es de gran interés en la industria.

5.2. Dónde se enmarca nuestro trabajo

En las secciones anteriores hemos definido qué es un FMS y hemos hecho hincapié en cuál es el significado de la palabra flexible, además de presentar las ventajas que conlleva el trabajar con este tipo de sistemas. Todo esto tiene como fin último mostrar que vamos a trabajar con un ejemplo real, necesario y perdurable, esto es, los FMS son útiles y necesarios, están implantados hoy en día en infinidad de empresas y se presentan como la alternativa de futuro más cierta. Evidentemente, es un campo de trabajo muy extenso por lo que vamos a intentar situar donde se enmarca nuestro trabajo actual.

Un FMS (de igual modo que cualquier sistema de fabricación aunque no sea flexible) está compuesto principalmente de dos subsistemas: El **subsistema físico**, que consiste en los recursos físicos (componentes hardware) tales como cintas transportadoras, robots, buffers, estaciones de trabajo, etc., y el **subsistema de control**, el cual determina cómo utilizar el subsistema físico para organizar y optimizar el proceso de producción.

Subsistema Físico

Nuestro trabajo, claramente, se enmarca dentro del subsistema de control, pero nuestra especificación deberá modelar los componentes físicos del sistema por lo que no viene mal una pequeña mirada a estos componentes.

1. Estaciones de trabajo.

   El equipamiento usado en la parte de procesado o ensamblaje dependerá del tipo de trabajo a realizar por el sistema. En un sistema diseñado para operaciones de mecanizado, principalmente usa maquinaria CNC (*Computer Numeric Control*), por ejemplo fresadoras, moldeadoras, taladradoras, etc. Suelen incluir una unidad de registro de entrada de piezas y una comunicación monitorizada entre el sistema informático y el operador. Todas las situaciones deben estar bajo control: palés mal posicionados, elementos extraños en zonas de trabajo de maquinaria, etc.

2. Sistemas de transporte y almacenaje de material.

   Son los responsables de los movimientos de los productos entre estaciones así como de su transporte desde/hacia la entrada y salida del sistema. Estos sistemas pueden variar mucho en componentes y configuración, dependiendo de las características del FMS y del material utilizado para la producción. Las funciones a desempeñar son:

   - Permitir un movimiento libre y aleatorio de los productos entre las estaciones.
   - Permitir varias configuraciones de productos en el transporte (módulos de palés, robots).
   - Almacenamiento temporal.
   - Facilitar los accesos para la carga y descarga.
   - Compatibilidad con el sistema de control por computador.

   Todos estos componentes pueden unirse en diferentes configuraciones y, como veremos más adelante, es de gran ayuda el disponer de una herramienta de análisis como **BTC** que nos permita hacer un análisis comparativo de prestaciones en distintas posibles configuraciones antes de la implantación de una de ellas. Las configuraciones habituales que encontramos en los FMS son: distribución en línea, en bucle, escalada, en campo abierto y centrada en robot.

3. Sistemas de control por computador.

   El FMS incluye un sistema informático distribuido conectado a las estaciones de trabajo, al sistema de transporte de material y a otros componentes hardware. El sistema posee un ordenador central y microcomputadores en las máquinas junto con el resto de componentes.

La misión del ordenador central es la de coordinar las actividades de los diferentes componentes para lograr un funcionamiento global estable del sistema. Las funciones de un sistema de control por computador se pueden agrupar en las siguientes categorías: control de la estación de trabajo, distribución de las instrucciones de control a las estaciones de trabajo, control de producción, control de tráfico, control de lanzadera, monitorización de piezas, control de las herramientas, control de la vida útil de las herramientas, control de rendimiento y diagnóstico.

Subsistema de Control (Implementación y Planificación)

La implementación de un FMS representa una gran inversión por parte de la compañía. Es importante que esta implementación vaya precedida por una minuciosa planificación y un buen diseño, realizando al mismo tiempo una buena gestión de todos los recursos.

La fase inicial de la planificación de los FMS debe centrarse en las piezas que se van a producir, solventando cuestiones como consideraciones sobre familias de piezas, necesidades de procesamiento, características físicas de las piezas o volumen de producción. Después de que estos parámetros hayan sido definidos, se puede continuar con el diseño del sistema. Los factores importantes a especificar en un FMS son los siguientes:

- Tipos de estaciones de trabajo: Los tipos de máquinas se determinan según las necesidades de procesamiento de los tipos de piezas.
- Variaciones en las rutas de procesamiento y distribución del FMS: La distribución del FMS que se elige en función de las variaciones de la ruta de proceso.
- Sistema de manejo del material: La selección del equipamiento para el manejo del material y la distribución del sistema están estrechamente relacionados, ya que el tipo de sistema de manejo limita, hasta cierto punto, la elección de dicha distribución. Además, este sistema debe considerar las características de las piezas que se van a procesar.
- *Work-in-process* (WIP) y capacidad de almacenaje: El nivel de WIP debe ser planificado con un valor lo bastante alto para que las estaciones no sufran inanición, pero con un nivel lo bastante bajo para no congestionar el sistema. La capacidad de almacenaje debe ser compatible con el nivel de WIP.

- Herramientas: Debe decidirse el tipo y el número de herramientas para cada una de las estaciones. Se debe tener en cuenta las herramientas comunes en las estaciones. Cuantas más herramientas de este tipo tengan las estaciones, mayor será la flexibilidad en las rutas del proceso.

- Fijaciones de palés: Debe decidirse el número de palés necesarios en el sistema. Las piezas que tengan una configuración y un tamaño muy distinto necesitan fijaciones diferentes.

Una vez que tenemos claro cuales son los factores a especificar aún nos queda saber cómo hacer esa especificación. Para este propósito vamos a considerar la celda de fabricación cuya distribución se muestra en la Figura 5.1.

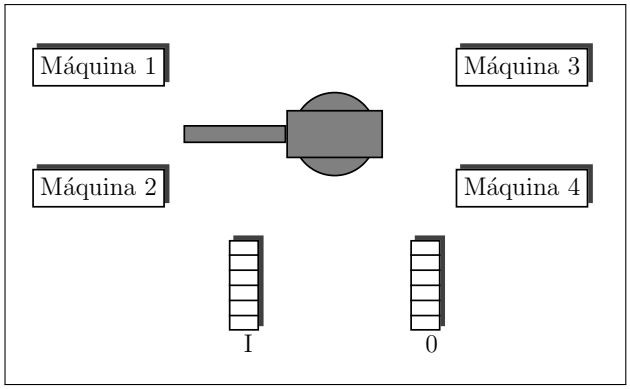


Figura 5.1: Sistema de Fabricación Flexible

La celda está compuesta por cuatro máquinas ($M1$, $M2$, $M3$ y $M4$) y un robot R responsable de cargar y descargar las máquinas así como de introducir las piezas en el sistema recogiéndolas de la cinta transportadora I y sacándolas de él a través de la cinta O. La máquina $M1$ puede procesar tres piezas a la vez, mientras que las máquinas $M2$, $M3$ y $M4$ sólo pueden procesar simultáneamente dos piezas.

En esta celda se pueden procesar dos tipos distintos de piezas. Las piezas del tipo A primero deben ser procesadas por la máquina $M1$ o $M3$ y luego pasar a $M2$ donde acaba su procesamiento. Las piezas del tipo B se procesan primero en $M1$, después en $M2$ y finalizan en $M4$. Para este primer ejemplo no entraremos en detalles sobre que tipo de procesamiento realiza cada máquina sobre las piezas.

Ya en este sencillo sistema podemos apreciar algunas características importantes que están presentes en casi todos los FMSs [ZD93]:

- **Es un sistema guiado por eventos**: El comportamiento del sistema se representa como un espacio de estados discretos donde un cambio en un estado ocurre al producirse una acción, por ejemplo una pieza nueva se incorpora o abandona la celda.

- **Es asíncrono**: Ciertas acciones en el sistema suceden de modo asíncrono, por ejemplo la finalización de la fase de procesamiento de una pieza en la máquina $M1$ es asíncrono con respecto a la carga de una nueva pieza en la máquina $M2$.

- **Existen relaciones secuenciales**: Ciertas acciones deben ocurrir de modo secuencial. Para que una pieza pueda ser descargada de la máquina $M1$, primero debe haber sido cargada en ella y haber finalizado su procesamiento.

- **Existe concurrencia**: El procesado de un pieza en $M1$ y otra en $M2$ se puede hacer concurrentemente y sin interacción entre ellas.

- **Existen conflictos**: Cuando el robot coge una pieza de tipo A es necesario tomar una decisión, ya que esta pieza puede ir a $M1$ o $M3$ (suponiendo que ambas tienen espacio para acogerla).

- **Existe no determinismo**: Como una consecuencia de los conflictos aparece cierto indeterminismo. En la situación anterior no podemos saber a priori que acción será tomada: la carga en $M1$ o en $M3$.

- **Existen bloqueos**: En el caso de que las tres máquinas estén ocupadas y el robot sostenga una pieza sin procesar con la intención de cargarla en alguna máquina, el sistema se encuentra en una situación de bloqueo total. No se puede ejecutar ninguna acción ya que ninguna máquina podrá ser descargada porque el robot ya que está ocupado y el robot no puede descargar la pieza que sostiene porque todas las máquinas están ocupadas.

- **Exclusión mutua**: El robot debe ser utilizado por varios procesos y parece razonable pensar que las acciones que esto conlleva se deban ejecutar en exclusión mutua.

Con esto podemos llegar a la conclusión de que el diseño de un FMS es una tarea realmente compleja: se deben combinar diferentes elementos y se tiene que tener en cuenta muchos aspectos. Esta complejidad da lugar a dos importantes necesidades:

1. El diseño de todo FMS debe hacerse de una manera jerárquica.
2. El uso de métodos formales es imprescindible para ser capaces de validar el sistema.

En cuanto a la primera de estas necesidades, existe una jerarquía clara en la que se define qué parte del diseño se realiza en cada nivel. Estos subsistemas de decisión incorporados en los FMS se resume bastante bien en [SV89] y pasamos a detallarlos brevemente para situar en qué nivel se encuentra nuestro trabajo actual. Así, los subsistemas de decisión se pueden dividir en los siguientes niveles:

- *Planning*. En este nivel se considera tanto la planta en su totalidad como la estimación de demanda. Se considera la producción a largo plazo, estableciendo el modo en el que los productos necesarios serán producidos durante un intervalo de tiempo.
- *Scheduling*. Bajando en la jerarquía, este nivel establece cuándo debe realizarse cada operación en cada producto.
- *Coordinación global*. Este nivel mantiene una visión completamente actualizada del estado del sistema y toma decisiones en tiempo real teniendo en cuenta el estado en el que se encuentran tanto los recursos del sistema como las piezas que están siendo procesadas.
- *Subsistema de coordinación*. El sistema de coordinación global puede ser descompuesto en módulos especializados para coordinar y supervisar subsistemas concretos como el de transporte (cintas transportadoras, el robot, un buffer, etc).
- *Control Local*. Este es el nivel más bajo de la jerarquía y es el encargado de la interacción con los sensores y otros componentes de bajo nivel.

Nuestro trabajo actualmente se enmarca en el nivel de **Coordinación Global** aunque no descartamos en un futuro extenderlo a los niveles de *Scheduling* y *Subsistema de Coordinación*.

La segunda necesidad que hemos visto imprescindible en el diseño de FMS es el uso de métodos formales. Según se recoge en [JMSW95] el uso de un marco formal en este campo tiene importantes beneficios:

1. En el proceso de formalizar los requerimientos del sistema ayudan considerablemente en la tarea de descubrir omisiones, ambigüedades y contradicciones.

2. Un método formal puede permitir que los desarrollos de los sistemas se realicen de una manera automática.

3. Los métodos matemáticos se pueden aplicar para verificar la corrección de los sistemas.

4. Una vez un subsistema ha sido verificado puede integrarse con toda confianza en sistemas más grandes.

5. Permiten hacer comparaciones entre distintos diseños para un mismo sistema.

Por otro lado, normalmente, los procesos de fabricación se clasifican en **continuos** (industrias químicas y petrolíferas, por ejemplo) y **discretos** (industrias de artículos para consumidores, ordenadores, etc.). De acuerdo al tipo de transformación que tiene lugar durante el proceso de producción, los sistemas discretos de producción se dividen en otras dos categorías dependiendo de si los procesos que llevan a cabo son "de ensamblaje" (**assemble**) o "de no ensamblaje" (**non-assembly**), esto es, de transformación. Los procesos de ensamblaje combinan varios componentes para obtener diferentes productos mientras que los que son de no ensamblaje se ocupan de la transformación (manipular, modelar, pintar, etc) la materia prima.

Con toda esta información podemos concluir que nuestro trabajo actual se sitúa en el **Subsistema de Control** aunque teniendo muy presente los elementos pertenecientes al Subsistema Físico (tanto en características como en lo referente a las distintas configuraciones a las que pueden optar). Dentro de la jerarquía de los subsistemas de decisión incorporados en los FMS, actualmente trabajamos en el nivel de **Coordinación Global** aunque es tan sólo como punto de partida ya que pretendemos extenderlo a otros niveles. Y sobre los procesos de fabricación con los que trataremos nos centramos en los procesos **discretos** pudiendo modelar tanto los de **ensamblaje** (assembly) como los de **no ensamblaje** (non-assembly).

5.3. Trabajos relacionados

El método de especificación formal que se ha utilizado principalmente para intentar modelar FMS son las Redes de Petri en sus distintas variantes. Pero antes de comentar un poco qué se ha hecho con ellas en este campo debemos destacar uno de los pocos lenguajes de especificación que se han utilizado en este campo: **PSF** (Process Specification Formalism), el cuál fue desarrollado en la Universidad de Amsterdam [MV90]. Este formalismo consigue dar una descripción exacta del comportamiento de los procesos tratándolos como bloques. Con ello pueden realizar su simulación permitiendo que se puedan chequear distintas especificaciones antes de ser implementadas. Pero tiene una carencia realmente importante; no es capaz de capturar características de tiempo real. Como consecuencia, propiedades de gran interés en los FMS como son la eficiencia (máxima productividad de las máquinas) o sincronizaciones no pueden ser modeladas.

Pero, sin lugar a duda, el método formal más ampliamente utilizado para modelar FMS son las Redes de Petri. Entre la amplia literatura dedicada a este tema podemos encontrar versiones que trabajan con una aproximación *top-down*, por ejemplo en [ZDD89], donde comenzando con una versión total del sistema se va refinando progresivamente, pero que habitualmente se ven en la necesidad de parar dicho refinamiento estando aún en niveles altos debido a la aparición de un problema bien conocido: la explosión de estados.

También podemos encontrar trabajos cuya forma de trabajar es *bottom-up* [KD91], esto es, comienzan con submodelos a los que les van integrando nuevos elementos, pero estos modelos serán más difíciles de validar principalmente por la aparición de bloqueos (*deadlocks*) al unir estos submodelos. En el camino intermedio entre estas dos metodologías encontramos también métodos híbridos con los que se ha intentado modelar FMS, por ejemplo en [PWX97].

Encontramos trabajos en los que se utilizan técnicas modulares para describir los FMS [DAJ95], obteniendo bloques de redes de Petri en los que se pueden estudiar ciertas propiedades de una manera más cómoda, pero que habitualmente encuentran problemas al unir los bloques e intentar que esas características estudiadas se conserven.

Nosotros trabajamos siguiendo la filosofía *bottom-up* y también utilizamos técnicas modulares, pero con una importante ventaja sobre los trabajos que lo hacen con Redes de Petri. Trabajamos con un álgebra de procesos donde la composicionalidad es precisamente una de sus características por lo

que nosotros podemos dividir nuestro sistema en subsistemas para especificarlos de una manera más cómoda y unirlos más tarde con la certeza de que las propiedades que tengan dichos subsistemas se mantendrán en el sistema formado por su unión.

En los trabajos preocupados por modelar FMS usando Redes de Petri son conscientes del problema que presenta la aparición de los bloqueos por lo que se ha dedicado un considerable esfuerzo a solucionarlo. Estos esfuerzos se pueden dividir principalmente en tres líneas: detección y recuperación de bloqueos (por ejemplo en [WNSS94]), evitar bloqueos ([Law99] entre otros) y prevención de bloqueos ([TGVCE98]).

En el campo donde mejores resultados se están obteniendo es en el de la prevención de bloqueos donde lo que se hace habitualmente es añadir al sistema una política para prevenir estos bloqueos. Pero el resultado que hemos visto en este tipo de trabajos es que con ello añaden nuevos elementos a la red lo que significa que el problema de la explosión de estados se incrementa todavía más viéndose entonces obligados a buscar soluciones para reducir dicha red. Como ejemplo de este problema podíamos citar, por ser de los más recientes, el trabajo de Uzam en [Uza02] donde se define una política óptima de prevención de bloqueos basada en el análisis de grafos de alcanzabilidad obtenidos a partir del modelo en redes de Petri del sistema al que se le ha añadido nuevos lugares y arcos para poder incluir la política propuesta (estos nuevos elementos se obtiene haciendo uso de la teoría de regiones, en este caso). Pero el mismo autor se ve en la necesidad de, más tarde, en [Uza04] proponer un forma de reducir estas redes de Petri porque, como él mismo comenta, las redes obtenidas para FMS grandes son realmente intratables.

Otra línea de trabajo que hemos encontrado para modelar FMS y que nos ha parecido muy interesante es la que se basa en crear y utilizar módulos predefinidos. En concreto en [MMR+98] se presenta un método llamado TRACE (más tarde fue ampliado en diversos trabajos) que usa bloques predefinidos de redes de Petri para modelar el comportamientos de los sistemas. Este método empieza haciendo una descripción informal del sistema que pasa a una especificación basada en trazas (siguiendo también el camino de CSP) antes de obtener la especificación con redes de Petri con bloques que más tarde podrán ser reutilizados.

Esta idea nos parece bastante buena ya que ciertas partes de las especificaciones de los FMS (sobre todo las que modelan sus componentes) son casi siempre iguales, por lo que sería de gran ayuda contar con *librerías* donde ya estuvieran definidas. Pero no creemos que sea necesario utilizar un lenguaje para especificar las trazas y más tarde pasarlo a redes de Petri. ¿No sería de

gran ayuda contar con un lenguaje que pudiera recoger toda la información? Ese lenguaje es **BTC**. A continuación mostraremos como es capaz de modelar los FMS y dejaremos para un estudio posterior la interesante idea de generar *bloques* que pueda ser reutilizados.

5.4. Especificación de FMSs

5.4.1. Metodología

Como ya ha quedado claro en las secciones anteriores los FMS presentan una complejidad y dimensiones importantes, por lo que se hace necesario abordar su diseño de una manera estructurada. Para ello dividimos el trabajo en cuatro fases. En la primera fase se identifica cada elemento individual del sistema y se establece cómo se especificará. En una segunda fase definimos para cada tipo de recurso un conjunto con las acciones que necesitarán para su ejecución al menos un recurso de ese tipo. En una tercera fase será necesario determinar de cuántos recursos de cada tipo dispone el sistema. Y en una cuarta y última fase nos quedará incorporar las trazas a los distintos procesos para obtener la especificación completa del sistema.

Empezamos recordando cuales son los componentes habituales en los Sistemas de Fabricación Flexible:

- Piezas (sin procesar).
- Estaciones de trabajo (nos referiremos a ellas como *máquinas* por comodidad).
- Sistemas de transporte: Cintas, Robots y Brazos articulados.
- Sistemas de almacenaje: Buffers (almacenaje temporal), módulos de palés.

Para cada pieza que entra en el sistema se arrancará un proceso que será el encargado de modelar las fases por las que la pieza debe pasar, usando los distintos recursos que necesite, hasta llegar al final de su procesamiento. Suponemos, sin pérdida de generalidad por ello, que tenemos en la entrada al sistema a modelar tantas piezas como se necesiten.

El resto de elementos de un FMS son todos considerados como recursos compartidos. Por lo tanto, debemos determinar qué tipo de recurso es cada

uno, qué acciones requieren de cada recurso, la duración de dichas acciones (en caso de que el recurso a utilizar sea expropiativo) y el número de procesos que pueden hacer uso de cada recurso simultáneamente. Sobre este último punto recordar que con nuestro lenguaje podemos simular paralelismo real, esto es que dos acciones que utilizan el mismo recurso pueden estar ejecutándose en el mismo instante de tiempo. Por esto, por ejemplo, si el sistema dispone de una máquina que puede pintar dos piezas al mismo tiempo se especificará como dos recursos que pintan cada uno una pieza.

Veamos la metodología que seguimos mediante un pequeño ejemplo. Supongamos el sistema mostrado en la Figura 5.2. Es un sistema discreto de transformación (discrete non-assembly system) encargado de pintar piezas. Consta de dos máquinas y un brazo articulado. El brazo articulado recoge una pieza de la entrada y la pasa por la máquina M_1 y después por M_2. En la máquina M_1 la pieza sufre dos *transformaciones*; primero es lavada y más tarde secada. Después en la máquina M_2 será pintada antes de abandonar el sistema.

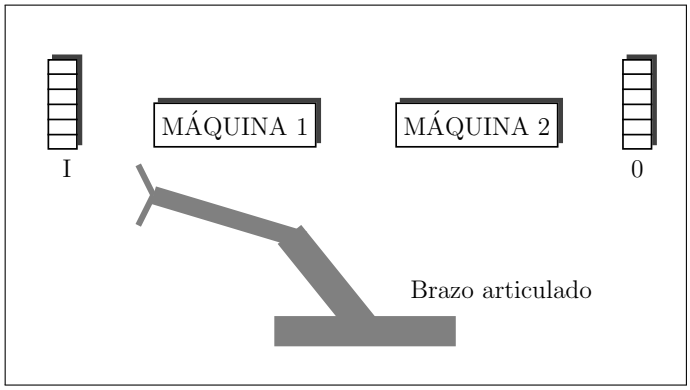


Figura 5.2: Celda de Fabricación Flexible Discreta de Transformación

En él, además de las piezas, que generarán cada una un proceso, tenemos tres recursos compartidos por dichos procesos. El brazo articulado (A_1) y las dos máquinas (M_1 y M_2). El primero será modelado como recurso no expropiativo mientras que las máquinas serán recursos expropiativos. Así, hemos identificado cada elemento del sistema y decidido como se especificará, con lo que hemos acabado la primera fase.

Pasamos a la segunda en la que tenemos que definir para cada tipo de recurso el conjunto de acciones que lo utilizan. Hemos dicho que el brazo articulado es un recurso no expropiativo (coge una pieza y no la suelta durante

todo el recorrido por el sistema) por lo que será necesario solicitarlo para poder utilizarlo y liberarlo cuando deje de ser necesario. Esto será especificado por las acciones r_arm y su conjugado ($\widehat{r\_arm}$).

Las máquinas son recursos expropiativos. La máquina M_1 es necesaria para realizar dos acciones; limpieza y secado, esto es $Z_1 = \{clean, dry\}$ que requieren 2 y 5 unidades de tiempo respectivamente. Para pintar la pieza se necesita la máquina M_2 ($Z_2 = \{paint\}$) y emplea 3 unidades de tiempo en esta acción. En una tercera fase necesitamos determinar el número de recursos de cada tipo disponibles en el sistema. En este ejemplo sencillo, suponemos que tenemos sólo un recurso de cada tipo (un brazo articulado y una máquina de cada tipo) y que cada uno puede trabajar con tan sólo una pieza a la vez ($N_1 = N_2 = N_3 = 1$).

Ya sólo nos queda la última fase para la especificación del sistema, hemos identificado cada uno de sus elementos, ahora nos queda la segunda fase que consiste en especificar la traza de cada proceso. En este caso dicha traza es bastante trivial y no requiere más explicación, con lo que la especificación del sistema quedaría como se muestra en la Figura 5.3.

$$\{[\text{Ex\_sys}]\}_{Z,\ \mathcal{N}} \equiv \{[piece_1 \parallel ... \parallel piece_i]\}_{\{\{clean,dry\},\{paint\},\{r\_arm\}\},\{1,1,1\}}$$

$$\begin{aligned} piece \equiv & < r\_arm, \mathcal{N} > . \\ & < clean, 2, \mathcal{N} > .< dry, 5, \mathcal{N} > . < paint, 3, \mathcal{N} > . \\ & < \widehat{r\_arm}, \mathcal{N} > \end{aligned}$$

Figura 5.3: Especificación Celda de Fabricación Flexible de la Figura 5.2

5.4.2. Caso de estudio

Una vez que hemos visto con un pequeño ejemplo cuál es la metodología que vamos a seguir a la hora de especificar FMS, pasemos a trabajar con un sistema más próximo a los reales en el que ya encontraremos todos los componentes típicos de los FMS. Como se comentó anteriormente, nuestra álgebra (**BTC**) es capaz de especificar fácilmente tanto Celdas de Fabricación Flexible Discretas de *Ensamblaje* como de *Transformación*. El sistema que hemos escogido para nuestro caso de estudio contendrá celdas de ambos tipos.

En dicho sistema, encontraremos distintos tipos de recursos, en concreto tendrá máquinas, robots, cintas transportadoras y buffers. Y por supuesto habrá que modelar adecuadamente acciones síncronas y asíncronas, relaciones secuenciales y concurrentes, no determinismo, exclusión mutua y por supuesto evitar posibles conflictos o bloqueos.

Nuestra celda consta de seis máquinas identificadas desde $M1$ hasta $M6$, cuatro robots ($R1$ - $R4$), dos buffers ($B1$ y $B2$), dos cintas transportadoras internas ($C1$ y $C2$) y dos cintas más para suministrar las piezas al sistema y extraerlas de él una vez que su procesamiento haya finalizado. Esta celda recibe dos tipos diferentes de piezas (A y B) las cuales son transformadas en las cinco primeras máquinas para más tarde ser ensambladas por la máquina $M6$. Qué tipo de transformación sufren las piezas en cada una de las máquina no nos interesa para la especificación, por lo que las identificaremos simplemente con acciones del tipo $process\_mx$ donde mx será la máquina necesaria para ejecutar esa acción. Un esquema de este sistema se puede encontrar en la Figura 5.4.

Las piezas de tipo A deben ser procesadas primero en la máquina $M1$ o en la máquina $M2$ y después pasar a la máquina $M3$. Más tarde, en la máquina $M6$, son ensambladas con las piezas parcialmente procesadas de tipo B. Por otro lado, las piezas de tipo B antes de ensamblarse con las de tipo A deben ser procesadas primero en la máquina $M4$ y después en $M5$.

El buffer $B1$ almacena temporalmente las piezas de tipo A provenientes de las máquinas $M1$ y $M2$ que esperan poder ser procesadas en la máquina $M3$. Y el buffer $B2$ almacena las piezas de tipo B que vienen de la máquina $M4$ y que deben ser tratadas en $M5$ antes de pasar a ensamblarse con las piezas de tipo A. Las piezas de tipo A son transportadas desde la máquina $M3$ a la máquina $M6$ a través de la cinta transportadora $C1$ y las de tipo B desde $M5$ a $M6$ por la cinta $C2$.

El robot $R1$ es el encargado de recoger las piezas de la cinta de entrada y distribuirlas entre las máquinas $M1$, $M2$ y $M4$. Los robots $R2$ y $R3$ son los encargados de manejar las entradas y salidas de los buffers $B1$ y $B2$ respectivamente: $R2$ descarga las máquinas $M1$ y $M2$, almacena las piezas de tipo A en el buffer $B1$ y luego las saca para llevarlas a la máquina $M3$. De forma similar trabaja el robot $R3$ con el buffer $B2$ y las máquinas $M4$ y $M5$. Finalmente el robot $R4$ recogen la pieza producida en la máquina $M6$ como resultado del ensamblaje de las piezas de tipo A y B y las coloca en la cinta de salida del sistema.

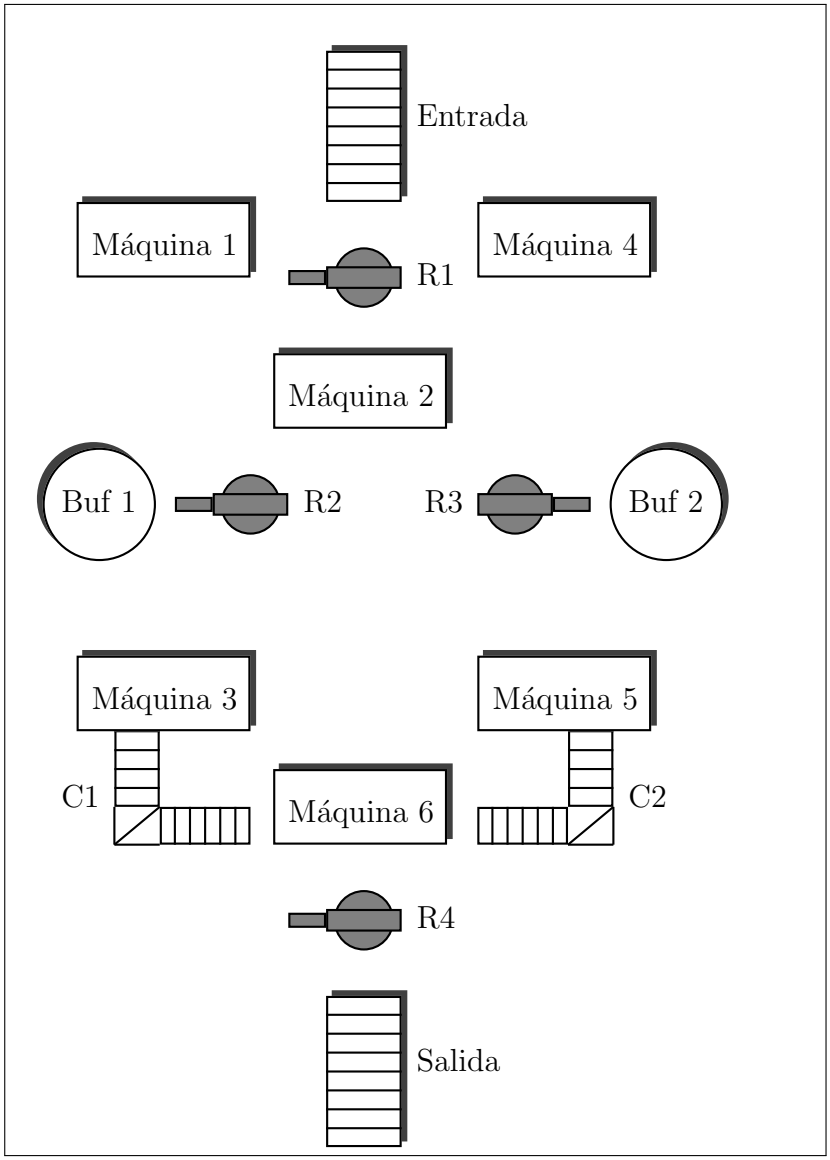


Figura 5.4: Celda de Fabricación Flexible a estudio

Siguiendo la metodología comentada en el apartado anterior, empezaremos identificando cada uno de los componentes de nuestro sistema. Las máquinas, los robots, los buffers y las cintas transportadoras serán recursos compartidos, y cada uno de ellos definirá un tipo de recurso en el sistema. Los recursos que modelan las máquinas serán recursos expropiativos mientras que el resto de recursos (los que modelan los robots, buffets y cintas transportadoras) serán no expropiativos.

Continuamos con la siguiente fase para lograr la especificación del sistema en la que tenemos que definir para cada tipo de proceso un conjunto con las acciones que necesitan al menos un recurso de ese tipo para su ejecución. Así para los recursos que modelan las máquinas Mi definimos los conjuntos:

$$Z_i = \{process\_mi\}$$

que incluyen las acciones ejecutadas en cada una de las máquinas. Ya hemos comentado anteriormente que no detallaremos en qué consiste cada una de esas acciones, pero sí es necesario determinar cuantas unidades de tiempo necesita cada una de ellas. Así, la acción $process\_m1$ requiere 5 unidades de tiempo, $process\_m2$ necesita 3 unidades, $process\_m3$ consume 4 unidades, $process\_m4$ dura 7 unidades, $process\_m5$ otras 8 unidades y finalmente $process\_m6$ que utiliza 3 unidades.

Para los recursos no expropiativos este conjunto de acciones es un poco diferente. Por ejemplo, un proceso necesita mantener un recurso *robot* (del tipo adecuado) durante todo el tiempo que dure la acción de mover una pieza sin que ningún otro proceso se lo pueda arrebatar en ese intervalo de tiempo. Por lo tanto será necesario pedirlo cuando se quiera utilizar y liberarlo cuando ya no sea necesario. Con esto nos aseguramos que se está utilizando en exclusión mutua. Así, las acciones que necesitan el recurso robot para ejecutarse serán la petición del recurso *robot* (r_ri) y su liberación que denotamos con su conjugado ($\widehat{r\_ri}$). Aunque, como ya se comentó, en el conjunto sólo incluiremos la petición ya que su conjugado se da por supuesto que está incluido, porque todo recurso no expropiativo necesita las dos acciones.

Esto mismo ocurrirá con el resto de recursos no expropiativos del sistema (los buffers y las cintas transportadoras) por lo que estamos en condiciones de definir los conjuntos de acciones correspondientes a cada tipo de recurso:

$$\begin{array}{llll} Z_7 = \{r\_b1\} & Z_9 = \{r\_c1\} & Z_{11} = \{r\_r1\} & Z_{13} = \{r\_r3\} \\ Z_8 = \{r\_b2\} & Z_{10} = \{r\_c2\} & Z_{12} = \{r\_r2\} & Z_{14} = \{r\_r4\} \end{array}$$

Ahora debemos conocer cuántos recursos de cada tipo hay en el sistema. Contamos con sólo un recurso de cada tipo de *robot*, una máquina de tipo $M1$ y otra de tipo $M2$, dos de tipo $M3$ y $M5$, cuatro de tipo $M4$ y tres de tipo $M6$. Para los recursos del tipo $buffer$ y $cinta\_transportadora$ el significado se puede asemejar a la capacidad que tienen. Los buffers $B1$ y $B2$ tienen unas capacidades de ocho y nueve piezas respectivamente y todas las cintas transportadoras cuentan con una capacidad de doce elementos. Con todo esto, obtenemos la información necesaria para conocer el conjunto $\mathcal{N}$ que quedaría de la siguiente manera:

$$\mathcal{N} = \{1, 1, 2, 4, 2, 3, 8, 9, 12, 12, 1, 1, 1, 1\}$$

Para cada pieza se generará un proceso cuya traza será la que se detalló en la descripción del ejemplo. Uniendo todo, obtenemos una especificación del sistema como la que se muestra en la Figura 5.5.

$$\{[\text{sys\_manu}]\}_{Z,\,\mathcal{N}} \equiv \{[GEN\_A \parallel GEN\_B \parallel ASSEMBLE]\}_{Z,\,\mathcal{N}}$$

$$GEN\_A \equiv (MAC\_1 \oplus MAC\_2) \parallel GEN\_A$$

$$MAC\_1 \equiv < r\_r1, \mathcal{N} >. < mov\_in\_m1, 2, \mathcal{N} > .< \widehat{r\_r1, \mathcal{N}} >.PROC\_A1$$

$$MAC\_2 \equiv < r\_r1, \mathcal{N} > . < mov\_in\_m2, 2, \mathcal{N} > .< \widehat{r\_r1, \mathcal{N}} >.PROC\_A2$$

$$GEN\_B \equiv < r\_r1, \mathcal{N} > . < mov\_in\_m4, 2, \mathcal{N} > .< \widehat{r\_r1, \mathcal{N}} >. (PROC\_B \parallel GEN\_B)$$

$$\begin{aligned} PROC\_A1 \equiv\ & < process\_m1, 5, \mathcal{N} >. < r\_b1, \mathcal{N} > . < r\_r2, \mathcal{N} > . \\ & < mov\_m1\_b1, 2, \mathcal{N} > .< \widehat{r\_r2, \mathcal{N}} >. < r\_r2, \mathcal{N} > . \\ & < mov\_b1\_m3, 2, \mathcal{N} > .< \widehat{r\_b1, \mathcal{N}} >.< \widehat{r\_r2, \mathcal{N}} >. \\ & < process\_m3, 4, \mathcal{N} > . < r\_c1, \mathcal{N} > .< syn\_A, \mathcal{N} >. \\ & < \widehat{r\_c1, \mathcal{N}} > \end{aligned}$$

$$\begin{aligned} PROC\_A2 \equiv\ & < process\_m2, 3, \mathcal{N} > .< r\_b1, \mathcal{N} >. < r\_r2, \mathcal{N} > . \\ & < mov\_m2\_b1, 2, \mathcal{N} > .< \widehat{r\_r2, \mathcal{N}} >. < r\_r2, \mathcal{N} > . \\ & < mov\_b1\_m3, 2, \mathcal{N} > .< \widehat{r\_b1, \mathcal{N}} >.< \widehat{r\_r2, \mathcal{N}} >. \\ & < process\_m3, 4, \mathcal{N} > . < r\_c1, \mathcal{N} > .< syn\_A, \mathcal{N} >. \\ & < \widehat{r\_c1, \mathcal{N}} > \end{aligned}$$

$$\begin{aligned} PROC\_B \equiv\ & < process\_m4, 7, \mathcal{N} > . < r\_b2, \mathcal{N} > . < r\_r3, \mathcal{N} > . \\ & < mov\_m4\_b2, 2, \mathcal{N} > .< \widehat{r\_r3, \mathcal{N}} >. < r\_r3, \mathcal{N} > . \\ & < mov\_b2\_m5, 2, \mathcal{N} > .< \widehat{r\_b2, \mathcal{N}} >.< \widehat{r\_r3, \mathcal{N}} >. \\ & < process\_m5, 8, \mathcal{N} > . < r\_c2, \mathcal{N} > .< syn\_B, \mathcal{N} >. \\ & < \widehat{r\_c2, \mathcal{N}} > \end{aligned}$$

$$\begin{aligned} ASSEMBLE \equiv\ & (< syn\_A, \mathcal{N} > \parallel < sync\_B, \mathcal{N} >). \\ & < process\_m6, 3, \mathcal{N} > . < r\_r4, \mathcal{N} > . \\ & < mov\_m6\_out, 2, \mathcal{N} > .< \widehat{r\_r4, \mathcal{N}} >.ASSEMBLE \end{aligned}$$

Figura 5.5: Especificación del FMS de la Figura 5.4

En esta especificación podemos apreciar que, a pesar de que el sistema que estamos modelando recoge todos los posibles elementos que pueden aparecer en un FMS real, la especificación sigue siendo fácilmente comprensible y no excesivamente grande.

En la especificación, hemos marcado con distintos colores algunas cosas que queremos comentar. En rojo hemos marcado las acciones de sincronización necesarias para que la máquina de ensamblaje no comience su trabajo hasta que no tenga disponibles una pieza de cada tipo. En verde hemos marcado una acción temporizada de las que se realizan en las máquinas. Para resaltar cómo se solicitan y liberan los recursos no expropiativos hemos utilizado el color azul en el caso de un robot, y el cyan en el caso de un buffer. Hemos marcado los dos casos ya que, a pesar de que tanto el buffer como el robot son recursos no expropiativos, su interpretación difiere un poco, ya que con un robot se modela la idea de apropiación de un recurso mientras que en el caso de los buffers lo que se modela es su capacidad.

5.5. Evaluación de prestaciones en un FMS

Una vez visto que somos capaces de especificar FMS complejos de una manera natural, sin necesidad de modelar cada recurso como un proceso, pasamos a realizar una evaluación de prestaciones sobre un FMS con el fin de mostrar qué conclusiones podemos obtener de ese análisis y para qué puede ser útil.

Para mostrar la utilidad de este análisis no es necesario trabajar con un sistema realmente grande, por lo que hemos optado por un tipo de celda de fabricación flexible no muy grande, pero que aparece en casi todos los FMS: una celda de ensamblaje (Figura 5.6). Dicha celda está formada por tres máquinas y un robot. La máquina $M1$ trabaja con piezas de tipo A y la máquina $M2$ con piezas de tipo B. Más tarde, cuando las piezas ya han sido tratadas por estas máquinas, pasan a la máquina $M3$ donde se ensamblará una pieza de tipo A con una de tipo B dando un nuevo tipo de pieza a la salida. El movimiento de las piezas en la celda la realiza un único robot que, evidentemente, trabaja en exclusión mutua.

Para hacer el estudio de una forma más cómoda, supondremos que cada máquina del sistema sólo puede estar trabajando con una pieza a la vez, aunque como se ha visto anteriormente, esto no es una restricción sino simplemente una simplificación para mayor comodidad.

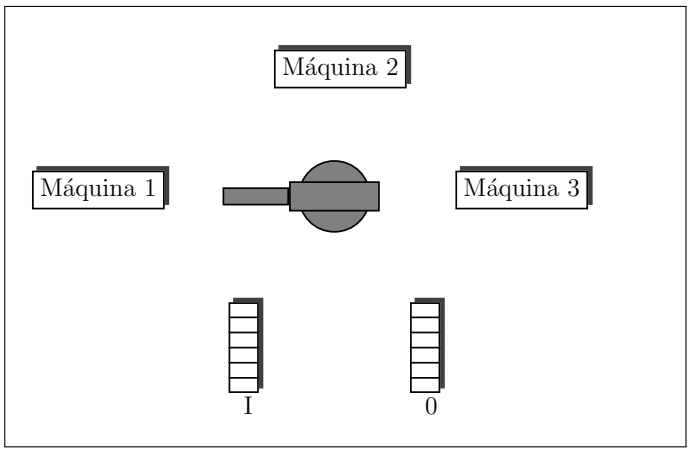


Figura 5.6: Celda Flexible de Ensamblaje

A pesar de parecer un ejemplo sencillo, recoge todas las características que hemos visto que habitualmente se encuentran en los FMS. Podemos encontrar relaciones secuenciales y concurrentes, existen conflictos que derivan en no determinismo, existe la posibilidad de que aparezcan bloqueos y cuenta con recursos que deben utilizarse en exclusión mutua. Además cuenta tanto con recursos expropiativos como no expropiativos y acciones de los tres tipos que hemos considerado (acciones con tiempo, acciones sin tiempo y acciones especiales).

Empecemos especificando el sistema. Encontramos cuatro recursos en él, tres máquinas que serán recursos expropiativos y para las que definiremos los siguientes conjuntos de acciones:

$$Z_1 = \{process\_m1\}$$
$$Z_2 = \{process\_m2\}$$
$$Z_3 = \{process\_m3\}$$

y un robot que será un recurso no expropiativo. Será necesario solicitarlo antes de su utilización y devolverlo al finalizar. Para el robot el conjunto de acciones que definimos será:

$$Z_4 = \{r\_robot\}$$

Para el estudio de prestaciones de este sistema, hemos considerado que podemos suponer que el sistema cuenta con un solo recurso de cada tipo sin perder generalidad, así:

$$\mathcal{N} = \{1, 1, 1, 1\}$$

Una vez cubiertas las fases previas de la especificación nos queda añadir la traza de los procesos, dando como resultado la especificación mostrada en la Figura 5.7.

$$Z_1 = \{process\_m1\} \quad Z_2 = \{process\_m2\} \quad Z_3 = \{process\_m3\} \quad Z_4 = \{r\_robot\}$$

$$\mathcal{N} = \{1,1,1,1\}$$

$$\{[\text{sys\_assembly\_cell}]\}_{Z,\,\mathcal{N}} \equiv \{[(GEN\_A \parallel GEN\_B) \parallel ASSEMBLE]\}_{Z,\,\mathcal{N}}$$

$$GEN\_A \equiv < r\_robot, \mathcal{N} > . < mov\_in\_m1, 2, \mathcal{N} > .< \widehat{r\_robot}, \mathcal{N} >. (PROC\_A \parallel GEN\_A)$$

$$GEN\_B \equiv < r\_robot, \mathcal{N} > . < mov\_in\_m2, 2, \mathcal{N} > .< \widehat{r\_robot}, \mathcal{N} >. (PROC\_B \parallel GEN\_B)$$

$$PROC\_A \equiv < process\_m1, 5, \mathcal{N} > . < r\_robot, \mathcal{N} > . < mov\_m1\_m3, 2, \mathcal{N} > . < \widehat{r\_robot}, \mathcal{N} >. < syn\_m1, \mathcal{N} >$$

$$PROC\_B \equiv < process\_m2, 7, \mathcal{N} > . < r\_robot, \mathcal{N} > . < mov\_m2\_m3, 2, \mathcal{N} > . < \widehat{r\_robot}, \mathcal{N} >. < syn\_m2, \mathcal{N} >$$

$$ASSEMBLE \equiv (< sync\_1, \mathcal{N} > \| < sync\_2, \mathcal{N} >). < process\_m3, 4, \mathcal{N} > . < r\_robot, \mathcal{N} > . < mov\_m3\_out, 2, \mathcal{N} > .< \widehat{r\_robot}, \mathcal{N} >. ASSEMBLE$$

Figura 5.7: Especificación de Celda de Ensamblaje

Haciendo uso de nuestra álgebra **BTC**, hemos conseguido dejar reflejado en esta especificación las características temporales del sistema, teniendo, además, en cuenta el número de recursos disponibles en el sistema en cada momento. Ahora nos enfrentamos a un problema de optimización de rendimiento concerniente a minimizar el tiempo máximo necesario para la finalización del procesamiento de un número de piezas.

Es evidente que la elección de este parámetro para su estudio se ha hecho por cuestiones económicas que, a fin de cuentas, son las razones de más peso para las empresas, donde cualquier pérdida de tiempo siempre estará suponiendo una pérdida económica.

Podemos enfocar nuestro trabajo en dos vías diferentes según las necesidades. Si lo que vamos a hacer es implantar un sistema nuevo, utilizaremos nuestra álgebra para especificar distintas configuraciones posibles. Más tarde usaremos el algoritmo de evaluación de prestaciones sobre esas especificaciones y podremos hacer un estudio comparativo de ellas, dejándonos la opción de escoger aquella que se ajuste mejor a los requerimientos temporales que deba cumplir el sistema haciendo uso del menor número de recursos (que también suponen un coste).

En el caso de que ya tengamos el sistema implantado, lo que hacemos es especificar la configuración actual que presenta y realiza un estudio de dónde se producen los mayores retrasos, dónde hay un cuello de botella, etc... de forma que podamos proponer nuevas configuraciones que presenten mejorías.

En el ejemplo que presentamos (celda de ensamblaje), empezamos suponiendo que ya tenemos el sistema implantado y especificado. Ahora haremos un estudio para ver si somos capaces de mejorar su rendimiento a un coste razonable. Partiendo de su especificación y haciendo uso de la semántica operacional, construimos su grafo de transición de estados sobre el que aplicaremos el algoritmo de evaluación de prestaciones. Este algoritmo, como se explicó cuando se presentó el álgebra de procesos en el Capítulo 3, calcula el tiempo mínimo para alcanzar un determinado estado, con lo que nos servirá para calcular cuál es el tiempo mínimo en procesar un determinado número de piezas o cuál es el número de piezas procesadas por unidad de tiempo (*throughput*).

Anteriormente ya se comentó que en los FMS los conflictos están presentes y que requieren que se tome decisiones al respecto si no queremos que desemboquen en un no determinismo a resolver por el propio sistema. En el ejemplo que nos ocupa, el recurso *Robot* es solicitado por distintos procesos y es frecuente que no puedan ser atendidas estas peticiones inmediatamente, por lo que puede haber más de una petición a la espera de ser atendidas. Para solucionar en la medida de lo posible este no determinismo y poder obtener resultados bajo las mismas condiciones con el fin de que puedan más tarde ser comparados, debemos de establecer unas estrategias de decisión. Nosotros empleamos una estrategia FCFS (First Come Firts Serve) para decidir el orden en el que las peticiones serán atendidas por el *Robot*. Puede ocurrir que en la misma unidad de tiempo, dos o más peticiones intenten entrar a la vez en esta cola de servicio del *Robot*, para que esta entrada tampoco sea no determinista debemos establecer unas prioridades para solucionar esta situación. Con estas premisas y haciendo uso del algoritmo de evaluación de prestaciones veamos que resultados obtenemos.

Empecemos asignando mayor prioridad al proceso B (el proceso que especifica el comportamiento de las piezas de tipo B). Con estas premisas, observamos que el sistema necesita 13.06 unidades de tiempo para procesas 100 piezas, eso es, obtenemos un throughput de 7.65 (el sistema puede procesar, en media, 7.65 piezas por unidad de tiempo). Pero cualquier cambio tanto en los requerimientos físicos como en las estrategias de decisión adoptadas puede hacer que este rendimiento varíe.

Así, podemos observar que con sólo cambiar la prioridad de los procesos para acceder a la cola de servicio del *Robot*, los resultados obtenidos son significativamente diferentes. Por ejemplo, si ahora decidimos que el proceso A tiene la mayor prioridad, obtenemos que el sistema tarda 11.16 unidades de tiempo en procesar 100 piezas con un throughput de 8.96, mientras que cuando la mayor prioridad es asignada al proceso *Assemble* el tiempo necesario para procesar el mismo número de piezas pasa a ser 11.08 (throughput de 9.02).

Con estos datos podemos observar que una pequeña modificación, tal como es variar la prioridad asignada a un proceso sólo para el caso de que solicite el uso del recurso *Robot* en el mismo instante de tiempo que otro proceso (prioridad para entrar en la cola FCFS), puede dar diferentes resultados, aún cuando estamos considerando una pequeña celda de tan sólo tres máquinas, y que el estudio ha sido realizado sólo para 100 piezas. Con esto podemos hacernos una idea de que una pequeña modificación en las decisiones de diseño establecidas pueden traer grandes consecuencias en sistemas reales cuyo tamaño es considerablemente superior, por lo que las ventajas de poder hacer una comparación y evaluación de los resultados obtenidos con distintas especificaciones queda claramente evidenciado.

Pero sigamos estudiando un poco más nuestro sistema. Si observamos el sistema podríamos pensar que el recurso compartido *Robot* es el causante de un cuello de botella en el sistema y que quizás el rendimiento mejoraría si tuviéramos dos recursos de este tipo. Para estudiar esta suposición, en la especificación cambiaríamos el valor del conjunto $\mathcal{N}$ aumentando hasta 2 el número de recursos de tipo *Robot* ($\mathcal{N} = \{1, 1, 1, 2\}$), generaríamos nuevamente el grafo de transición de estados y usaríamos el algoritmo de evaluación de rendimiento para descubrir si realmente este cambio merece la pena.

Para poder hacer una comparación realista con los resultados obtenidos previamente, obtenemos el throughput del sistema para las mismas tres variaciones que hemos visto en el supuesto de contar con un solo recurso *Robot* (mayor prioridad al entrar en la cola FCFS para el proceso A, el B o el *Assemble*) y descubrimos que cuando el sistema cuenta con 2 recursos *Robot*

esta decisión no influye demasiado, ya que en los tres casos el tiempo que necesita el sistema para procesar 100 piezas es de 11.06 unidades de tiempo con un throughput de 9.04.

El hecho de que en este caso se obtengan los mismos resultados tiene bastante sentido, ya que al haber más recursos del tipo *Robot*, las colisiones para entrar a las distintas colas de servicio de esos recursos serán menores y no habrá que hacer uso de dicha estrategia de decisión con tanta frecuencia, y su influencia en los resultados finales disminuirá considerablemente.

Pero comparémoslo ahora con los datos obtenidos cuando el sistema sólo contaba con un recurso *Robot* para lo cual puede ayudar ver la Tabla 5.2 donde hemos recogido estos resultados.

	1 Robot			**2 Robots**		
Prioridad —->	**Proceso A**	**Proceso B**	**Proceso ASSEMBLE**	**Proceso A**	**Proceso B**	**Proceso ASSEMBLE**
Tiempo para procesar 100 piezas	11,16	13,06	11,08	11,06	11,06	11,06
Piezas por unidad de tiempo (Throghput)	8,96	7,65	9,02	9,04	9,04	9,04

Cuadro 5.2: Resultados para procesar 100 piezas en la Celda de ensamblaje

Podemos observar que el throughput del sistema cuando tiene dos robots (9.04) está muy próximo al que se consigue con sólo un robot (9.02) en el caso de que la estrategia de decisión elegida sea la correcta (en este caso priorizar el proceso $ASSEMBLE$ en la entrada a la cola de servicio del recurso *Robot*). Con sólo este criterio parece claro que no es necesario (o recomendable) la inclusión de un nuevo robot, pero será el diseñador del sistema el que deba tomar la decisión ya que, evidentemente, habrá más parámetros a considerar en el diseño del sistema. En cualquier caso, queda claro que presenta una gran ventaja el hecho de poder simular el comportamiento del sistema antes de ser implantado.

Veamos más resultados que pueden ser interesantes. En el mercado existen distintos tipos de robots que presentan diferencias en las características técnicas. Hemos podido comprobar que para una misma funcionalidad existen distintos tipos de robots donde la mayor diferencia radica en la velocidad de sus movimientos y, como es de suponer, en el coste económico. Veamos que sucede si el robot de nuestro sistema es reemplazado por uno más económico pero que invierte en sus desplazamientos 9 unidades de tiempo.

La especificación anterior sería la misma ya que el sistema presenta el mismo comportamiento, pero habría que modificar los tiempos empleados en ejecutar las acciones temporizadas que requieren el uso del robot. Esto es, las acciones *mov_in_m*1, *mov_in_m*2, *mov_m*1_*m*3, *mov_m*2_*m*3 y *mov_m*3_*out* pasarían a tener una duración de 9 unidades de tiempo. Volvemos a hacer uso del algoritmo de evaluación de prestaciones y obtenemos que el sistema procesa 100 piezas en 45.18 unidades de tiempo cuando cuenta con sólo un robot (throughput de 2.21). Además, hemos podido observar que con los nuevos datos del sistema la política de decisión que antes vimos que tenía bastante importancia a la hora de tomar una decisión, ahora no es necesaria, ya que experimentalmente se ha comprobado que no existen colisiones entre los procesos a la hora de entrar en la cola de servicio del recurso *Robot*.

Es claro que el throughput del sistema ha disminuido considerablemente al contar con un robot mucho más lento (y seguramente también de menor coste) como en principio cabía de esperar, pero continuamos nuestro estudio calculando el throughput del sistema en el caso de que éste contara con dos recursos *Robots* de este tipo. Obtenemos un throughput de 4.00, lo que significa que el nuevo sistema invierte 24.97 unidades de tiempo en procesar 100 piezas.

Si comparamos los resultados obtenidos cuando este sistema cuenta con uno o dos recursos *Robots* (Tabla 5.3) parece evidente que las restricciones económicas deberían ser muy grandes para que el diseñador no se decida por el uso de dos robots, ya que, contando con dos robots, el número de piezas que podría procesar por unidad de tiempo sería casi el doble que cuando cuenta tan solo con uno teniendo el mismo número del resto de máquinas involucradas en esta celda.

	1 Robot	**2 Robots**
Tiempo para procesar 100 piezas	45,18	24,97
Throughput	2,21	4,00

Cuadro 5.3: Resultados para Robot Lento en la Celda de Ensamblaje

Algo que con nuestro estudio no podríamos decir con tanta seguridad es si vale la pena la inversión en el robot más rápido o no. Según los datos que nosotros hemos obtenido no cabría duda, mientras que con un robot rápido

(tarda 2 unidades de tiempo en sus desplazamientos) el sistema es capaz de procesar 9.02 piezas en el mejor de los casos, con el robot lento (utiliza 9 unidades de tiempo al moverse) sólo es capaz de procesar 2.21 piezas. Pero nos falta un dato de sumo interés en este caso; los costes de estos robots. Según el estudio que hemos realizado la variación en los costes puede ser realmente considerable, por lo que será el diseñador del sistema el que con todos los parámetros deba tomar esta decisión.

Capítulo 6

Conclusiones y Trabajo Futuro

En esta tesis hemos presentado un álgebra de procesos temporizada, denominada **BTC**, capaz de tomar en consideración el número y tipo de recursos disponibles en el sistema en cada momento. Para ello ha sido necesario aumentar la expresividad del lenguaje de especificación CSP con el fin de poder introducir aspectos cuantitativos en la fase de diseño que más tarde serán de utilidad al realizar el análisis de prestaciones.

Este nuevo lenguaje es capaz de recoger la existencia de distintos tipos de recursos en el sistema (trabaja con recursos heterogéneos), diferenciando entre el comportamiento de los que son expropiativos y los que no lo son. También recoge la duración de las acciones en un dominio discreto, con lo que, además, se puede expresar el concepto de consumo de recurso.

El hecho de tener en cuenta la disponibilidad de los recursos del sistema nos lleva a un caso particular de especial importancia: cuando el recurso compartido es el procesador, ya que con ello nos estamos adentrando en la teoría de scheduling al ser planificación lo que realmente estamos realizando.

De hecho, una ventaja fundamental de este lenguaje radica en el hecho de que es capaz de trabajar tanto en paralelismo real como en interleaving, dependiendo del número de procesadores con que se cuente al hacer la especificación de un sistema.

En esta tesis, también se ha presentado un análisis temporal de prestaciones haciendo uso de un grafo de transiciones, con el que se pretende maximizar el rendimiento de los sistemas al minimizar el tiempo necesario para alcanzar el estado final a partir del inicial.

Como casos de estudio reales hemos presentado la especificación y análisis de prestaciones del protocolo SET y de distintos sistemas de fabricación flexible.

En el protocolo SET hemos especificado los distintos protocolos de las dos fases que lo componen, aunque hemos visto que donde más interés puede tener el análisis de prestaciones que realizamos es en la fase de compra, por lo que es ahí donde hemos centrado nuestro interés. En esta fase se han hecho dos estudios distintos. Primero, teniendo en cuenta diferentes estimaciones para el número de compradores (en media) presentes en el sistema en un determinado momento, se ha obtenido el número idóneo de vendedores que serían necesario para cumplir con unos ciertos requerimientos temporales relativos al tiempo de servicio del cliente. Por otro lado, para intentar adaptarse a esos requerimientos temporales, se ha analizado la posibilidad contraria, esto es, si el número de vendedores en el sistema se mantiene fijo, calculamos cuantos compradores pueden aceptarse simultáneamente.

Respecto al trabajo realizado en los sistemas de fabricación flexible podemos decir que los resultados obtenidos son realmente interesantes. No sólo hemos constatado que **BTC** se adapta perfectamente a este tipo de sistemas, sino que, además, los resultados obtenidos tras su análisis de prestaciones son de gran utilidad al ser sistemas que se encuentran en continuo cambio ya que, como se comentó en el capítulo 5, tanto la configuración de un FMS como los elementos que lo constituyen sufren modificaciones constantes para poder adaptarse a las nuevas demandas de los usuarios.

En el estudio realizado, al utilizar el algoritmo propuesto de análisis de prestaciones, lo que se hace es calcular el tiempo mínimo en procesar un determinado número de piezas o cuál es el número de piezas procesadas por unidad de tiempo (*throughput*) que, de igual modo que el lenguaje, cumple perfectamente con las expectativas de información que el diseñador de este tipo de sistema necesita.

Hemos enfocado nuestro trabajo de análisis del rendimiento de los FMS en dos vías diferentes según las necesidades. Si lo que se va a hacer es implantar un sistema nuevo, utilizamos nuestra álgebra para especificar distintas configuraciones posibles. Más tarde usaremos el algoritmo de evaluación de prestaciones sobre esas especificaciones y podremos hacer un estudio comparativo de ellas, dejándonos la opción de escoger aquella que se ajuste mejor a los requerimientos temporales que deba cumplir el sistema, haciendo uso del menor número de recursos posibles (con el fin de disminuir los costes).

En el caso de que ya tengamos el sistema implantado, lo que hacemos es especificar la configuración actual que presenta y realiza un estudio de dónde se producen los mayores retrasos, dónde hay un cuello de botella, etc... de forma que podamos proponer nuevas configuraciones, quizás la necesidad de sustituir un elemento por otro o, en general, cualquier modificación que conlleve una mejoría. En este aspecto se han presentado estudios comparativos modificando las políticas de toma de decisión y las velocidades de los robots.

Para finalizar, diremos que pensamos que nuestro lenguaje **BTC** es capaz de modelar casi cualquier tipo de sistema real de una manera bastante intuitiva y sin necesidad de echar mano a ningún subterfugio que, habitualmente, terminan complicando las especificaciones y alejándose de comportamiento real del sistema. El análisis de prestaciones realizado nos arroja suficiente información para ayudar en la toma de decisiones necesarias a la hora de implantar o mejorar sistemas. Y dentro de los casos de estudio tratados, en los FMS hemos encontrado un interesante campo de actuación con el que tenemos intención de seguir trabajando, como se comentará más adelante en el apartado 6.2.

6.1. Publicaciones

El trabajo desarrollado en esta tesis ha dado como resultado siete artículos que han sido presentados en diversos congresos, uno nacional y el resto internacionales.

En las XII Jornadas de Concurrencia y Sistemas Distribuidos hicimos una primera presentación [RCCP04] de **BTC** en la que se tomaba en consideración los recursos existentes en el sistema (sólo recursos homogéneos) y que permitía trabajar tanto con paralelismo real como con interleaving. Pronto le añadimos el análisis de prestaciones con el que hemos trabajado en esta tesis y fue presentado en el *1st European Performance Engineering Workshop (EPEW'04)* de la *International Conference on Formal Techniques for Networked and Distributed Systems (FORTE'04)* [RCC+04].

Después de haber trabajado con unos sistemas que nos sirvieron como ejemplos de prueba, pasamos a utilizar el álgebra **BTC** ya en un sistema real. Así, realizamos la especificación y el análisis de prestaciones de la Fase de Compra del protocolo SET [RCCP05] que fue presentada en *Computer-Aided Verification of Information Systems (CAVIS'05)*. Más tarde se amplió nuestra algebra con el fin de poder trabajar con recursos heterogéneos y se hizo un estudió más detallado del protocolo SET [RCCP06a] que fue presentado en el *21st ACM Symposium on Applied Computing (SAC'06)*.

La versión de **BTC** en la que se incluyeron los recursos expropiativos y no expropiativos [RCCP06b] fue presentada en la *5th International Workshop on Automated Verification of Infinite-State Systems (AVIS'06)* donde se centró la atención en su utilidad para la verificación.

El segundo caso de estudio presentado en esta tesis, los sistemas de fabricación flexibles, fue el objeto dos artículos más; en el primero se realizó la especificación de un sistema real de fabricación flexible [RCCP06c] y se presentó en el *8th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC'06)*. En el último trabajo presentado [RCCP06d] se realizó un exhaustivo análisis de prestaciones sobre un FMS y se presentó en la conferencia internacional de *Computational Intelligence for Modelling, Control and Automation (CIMCA/IAWTIC'06)*.

6.2. Trabajo futuro

El álgebra que hemos definido nos parece suficientemente potente, por lo que nuestro trabajo futuro va encaminado hacia su mejor adaptación a sistemas reales, esto es, poder automatizar, en la medida de lo posible, tanto la realización de las especificaciones como, sobre todo, la realización del análisis de prestaciones. Así, podemos resumir que los puntos en los que pretendemos seguir trabajando son los siguientes:

- Nos parece clara la necesidad de diseñar una herramienta para generar el grafo de transición de estados a partir de la especificación de una manera automática, por lo que ese será uno de nuestros primeros objetivos.

- Además, pretendemos poder adaptarnos a alguna herramienta que sea capaz de realizar los cálculos necesarios en nuestro algoritmo de análisis de prestaciones. Si esto no fuera posible, deberemos desarrollarla nosotros.

- Dicho grafo de transición de estados, al modelar sistemas reales no triviales, adquiere unas dimensiones considerables, por lo que, como trabajo a realizar, nos planteamos el diseñar técnicas de reducción de estados. En este punto ya hemos hecho algún avance, descubriendo que con tan solo eliminar nodos equivalentes la mejoría es notoria.

- Dentro de la jerarquía de los subsistemas de decisión incorporados en los FMS, actualmente trabajamos en el nivel de Coordinación Global (*Global Coordination*), aunque es tan sólo como punto de partida, ya que pretendemos extenderlo a los niveles de Planificación (*Scheduling*) y Subsistema de Coordinación (*Subsystem Coordination*).

- Al trabajar con FMS hemos descubierto que ciertas configuraciones y comportamientos se repiten con frecuencia, por lo que, haciendo uso de la modularidad que nos ofrece nuestra álgebra de procesos, se nos ocurre el crear algo parecido a unas librerías que contengan módulos predefinidos, con lo que se facilitará considerablemente la especificación de sistemas grandes.

Y como trabajo más ambicioso y el que sería nuestro fin último, nos gustaría aplicar todo este trabajo en una fábrica donde se trabaje con FMS y poder colaborar en el diseño de sus sistemas.

Bibliografía

[ABC+94] M. Ajmone Marsan, A. Bianco, L. Ciminiera, R. Sisto, and A. Valenzano. A LOTOS extension for the performance analysis of distributed systems. *IEEE/ACM Transactions on Networking*, 2(2):151–165, 1994.

[AD94] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[BAL02] Mikael Buchholtz, Jacob Andersen, and Hans Henrik Løvengreen. Towards a Process Algebra for Shared Processors. *Electronic Notes in Theoretical Computer Science*, 52, 2002.

[BBS95] J.C.M. Baeten, J.A. Bergstra, and S.A. Smolka. Axiomatizing probabilistic processes: ACP with generative probabilities. *Information and Computation*, 121:234–255, 1995.

[BCHK93] Gérard Boudol, Ilaria Castellani, Matthew Hennessy, and Astrid Kiehn. Observing localities. *Theor. Comput. Sci.*, 114(1):31–61, 1993.

[BG98] M. Bernardo and R. Gorrieri. A tutorial on EMPA: A theory of concurrent processes with nondeterminism, priorities, probabilities and time. *Theoretical Computer Science*, 202:1–54, 1998.

[BG02] M. Bravetti and R. Gorrieri. The theory of interactive generalized semi-markov processes. *Theoretical Computer Science*, 282(1):5–32, 2002.

[BGL97] P. Brémond-Grégoire and I. Lee. A Process Algebra of Communicating Shared Resources with Dense Time and Priorities. *Theoretical Computer Science*, 189(1-2):179–219, 1997.

[BL91] Tommaso Bolognesi and Ferdinando Lucidi. Timed process algebras with urgent interactions and a unique powerful binary operator. In *REX Workshop*, pages 124–148, 1991.

[BM01] J.C.M. Baeten and C.A. Middelburg. Process algebra with timing: Real time and discrete time. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of process algebra*, chapter 10. North Holland, 2001.

[BMP02] G. Bella, F. Massacci, and L.C. Paulson. The verification of an industrial payment protocol: The SET purchase phase. In *Proceedings of 9th ACM Conference on Computer and Communications Security*, pages 12–20, 2002.

[BMP03] G. Bella, F. Massacci, and L.C. Paulson. Verifying the SET registration protocols. In *Proceedings of IEEE J. of Selected Areas in Communications*, pages 21(1):77–87, 2003.

[BP98] G. Bella and L.C. Paulson. Kerberos version IV: Inductive analysis of the secrecy goals. In *Proceedings of the Fifth European Symposium on Research in computer Security.*, pages 361–375. IEEE Computer Society Press, 1998.

[BPS01] J.A. Bergstra, A. Ponse, and S.A. Smolka, editors. *Handbook of Process Algebra*. Elsevier, 2001.

[Bro83] S.D. Brookes. *A Model for Communicating Sequential Processes*. PhD thesis, Oxford University, 1983.

[BW90] J.C.M. Baeten and W.P. Weijland. *Process Algebra*. Cambridge Tracts in Computer Science 18. Cambridge University Press, 1990.

[CCV+03] D. Cazorla, F. Cuartero, V. Valero, F.L. Pelayo, and J.J. Pardo. Algebraic theory of probabilistic and non-deterministic processes. *Journal of Logic and Algebraic Programming*, 55(1–2):57–103, 2003.

[DAJ95] Alan A. Desrochers and Robert Y. Al-Jaar. *Applications of Petri Nets in Manufacturing Systems. Modeling, Control and Performance Analysis*. IEEE Press, 1995.

[DP95] Pierpaolo Degano and Corrado Priami. Causality for mobile processes. In *ICALP*, pages 660–671, 1995.

[DY83] D. Dolev and A. Yao. On the Security of Public key Protocols. In *IEEE Transactions on Information Theory*, pages 29(2):198–208, March 1983.

[FKK96] A.O. Freier, P. Karlton, and P.C. Kocher. The SSL protocol v3.0. *Internet Draft*, 1996.

[GHR93] N. Götz, U. Herzog, and M. Rettelbach. Multiprocessor and distributed system design: The integration of functional specification and performance analysis using stochastic process algebras. In *16th Int. Symp. on Computer Performance Modelling, Measurement and Evaluation (PERFORMANCE'93), LNCS 729*, pages 121–146. Springer, 1993.

[Gru96] Damas P. Gruska. Process Algebra for Limited Parallelism. In *Proc. of Concurrency, Specification and Programming (CS&P'96)*, pages 61–74, 1996.

[Gru97] Damas P. Gruska. Bounded Concurrency. In *In Chlebus, B.S., Czaja, L., eds.: Proc. of 11th International Symposium on Fundamentals of Computation Theory (FCT'97), LNCS 1279, Springer*, pages 198–209, 1997.

[GSS95] R. van Glabbeek, S.A. Smolka, and B. Steffen. Reactive, generative and stratified models of probabilistic processes. *Information and Computation*, 121(1):59–80, 1995.

[GSST90] R. van Glabbeek, S.A. Smolka, B. Steffen, and C.M.N. Tofts. Reactive, generative, and stratified models of probabilistic processes. In *5th IEEE Symposium on Logic In Computer Science*, pages 130–141. IEEE Computer Society Press, 1990.

[HHK02] H. Hermanns, U. Herzog, and J-P. Katoen. Process algebra for performance evaluation. *Theoretical Computer Science*, 274(1–2):43–86, 2002.

[Hil96] J. Hillston. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.

[HKL+98] C. Heitmeyer, J. Kirby, B. Labaw, M. Archer, and R. Bharadwaj. Using Abstraction and Model Checking to Detect Safety Violations in Requirements Specifications. *IEEE Transactions on Software Engineering*, 24:927–948, 1998.

[Hoa78] C.A.R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[HR98] J. Hillston and M. Ribaudo. Stochastic process algebras: a new approach to performance modeling. In K. Bagchi and G. Zobrist, editors, *Modeling and Simulation of Advanced Computer Systems*, 1998.

[HS00] P.G. Harrison and B. Strulo. SPADES – a process algebra for discrete event simulation. *Journal of Logic Computation*, 10(1):3–42, 2000.

[HTWW96] N. Heintze, J. D. Tygar, J. Wing, and H. C. Wong. Model Checking Electronic Commerce Protocols. pages 147–164, 1996.

[JMSW95] S.B. Joshi, E.G. Mettala, J.S. Smith, and R.A. Wysk. Formal Models for Control of Flexible Manufacturing Cells: Physical and System Mode. *IEEE Transactions on Robotics and Automation*, 11:558–570, 1995.

[JYL01] B. Jonsson, W. Yi, and K.G. Larsen. Probabilistic extensions of process algebras. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of process algebra*, chapter 11. North Holland, 2001.

[Kai95] R. Kailar. Reasoning about accountability in proticols for electronic commerce. In *Proceedings of the 14th IEEE Symposium on Security and Privacy*, pages 236–250, 1995.

[KCS02] S. Kalyanasundaram, E.K.P. Chong, and N.B. Shroff. Optimal resource allocation in multi-class networks with user-specified utility functions. *Computer Networks*, 38:613–630, 2002.

[KD91] I. Koh and F. Dicesare. Modular transformation methods for generalizad Petri Nets and their application to automated Manufacturing Systems. *IEEE Trans. Syst., Man Cybern*, 21:1512–1522, 1991.

[KN98] V. Kessler and H. Neumann. A sound logic for analysing electronic commerce protocols. volume 1485 of *Lecture Notes in Computer Science*, pages 345–360. Springer, 1998.

[Law99] M. Lawley. Deadlock avoidance for production systems with Flexible Routing. *IEEE Trans. Robot. Automat.*, 15(3):497–409, 1999.

[LBGG94] I. Lee, P. Brémond-Grégoire, and R. Gerber. A Process Algebraic Approach to the Specification and Analysis of Resource-Bound Real-Time Systems, 1994.

[LCK+01] Insup Lee, Jin-Young Choi, Hee-Hwan Kwak, Anna Philippou, and Oleg Sokolsky. A Family of Resource-Bound Real-Time Process Algebras. In *Proc. of FORTE'01*, pages 443–458. Kluwer Academic Publishers, 2001.

[LdF99] L. Llana and D. de Frutos. Relating may and must testing semantics for discrete timed process algebras. In *ASIAN'99, LNCS 1742*, pages 74–86. Springer, 1999.

[LH01] G. Lowe and M.L. Hui. Fault-preserving simplifying transformations for security protocols. *Journal of Computer Security*, pages 9(3–46), 2001.

[Liu00] Jane W.S. Liu. *Real-Time Systems*. Prentice Hall, 2000.

[LN01] N. López and M. Núñez. A testing theory for generally distributed stochastic processes. In *CONCUR 2001*, pages 321–335. Springer, 2001.

[LNR02] N. López, M. Núñez, and F. Rubio. Stochastic process algebras meet Eden. In *Integrated Formal Methods 2002, LNCS 2335*, pages 29–48. Springer, 2002.

[Low96] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using CSP and FDR. volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer, 1996.

[LR92] D. Longley and S. Rigby. An automatic search for security flaws in key management schemes. In *Computers and Security*, pages 11(1):75–90, 1992.

[LS91] K.G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, 1991.

[MCF87] K.K. Millen, S.C. Clark, and S.B. Freedman. The Interrogator: protocol security analysis. In *IEEE transactions on Software Engineering*, pages SE–13(2), 1987.

[Mea99] C. Meadows. Inductive analysis of the Internet protocol TLS. In *Transactions on Information and System Security*, pages 2(3):332–351. ACM, 1999.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[Mil99] R. Milner. *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, 1999.

[MMR+98] A. Mazzeo, N. Mazzocca, S. Russo, C. Savy, and V. Vittorini. Formal specification of concurrent systems: a structured approach. *The Computer Journal*, 41(3):145–162, 1998.

[MS98] C. Meadows and P. Syverson. A formal specification of requirements for payment transactions in the SET protocol. In *Financial Cryptography*, pages 122–140, 1998.

[MSS98] J. Mitchell, V. Shmatikov, and U. Stern. Finite-state analysis of SSL 3.0. In *Proc. of the Seventh USENIX Security Symposium*, pages 201–216, 1998.

[MT90] F. Moller and C. Tofts. A temporal calculus of communicating systems. In *CONCUR'90, LNCS 458*, pages 401–415, Amsterdam, 1990.

[MV90] S. Mauw and G.J. Veltink. A process specification formalism. In *Fundamenta Informaticae XIII*, pages 85–139, 1990.

[MY92] Ugo Montanari and Daniel Yankelevich. A parametric approach to localities. In *ICALP*, pages 617–628, 1992.

[NR01] M. Núñez and I. Rodríguez. `PAMR`: A Process Algebra for the Management of Resources in Concurrent Systems. In *Proc. of FORTE'01*, pages 169–185. Kluwer Academic Publishers, 2001.

[NS91] X. Nicollin and J. Sifakis. An overview and synthesis on timed process algebras. In *Computer Aided Verification'91, LNCS 575*, pages 376–398, 1991.

[Núñ03] M. Núñez. Algebraic theory of probabilistic processes. *Journal of Logic and Algebraic Programming*, 56(1–2):117–177, 2003.

[Pau98] L.C. Paulson. Finite-state analysis of SSL 3.0. In *Proceedings of the Seventh USENIX Security Symposium*, pages 201–216, 1998.

[Plo81] G.D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Computer Science Department, Aarhus University, 1981.

[PWX97] J. Proth, L. Wang, and X. Xie. A class of Petri Nets for Manufacturing System integration. *IEEE Trans. on Robotics and Automation*, 13:317–326, 1997.

[QdFA93] J. Quemada, D. de Frutos, and A. Azcorra. TIC: A TImed Calculus. *Formal Aspects of Computing*, 5:224–252, 1993.

[RCC+04] M. Carmen Ruiz, Diego Cazorla, Fernando Cuartero, Juan José Pardo, and Hermenegilda Macià. A bounded true concurrency process algebra for performance evaluation. In *FORTE Workshops (EPEW'04), LNCS 3236*, Toledo, Spain, October 2004. Springer.

[RCCP04] M. Carmen Ruiz, Diego Cazorla, Fernando Cuartero, and Juan José Pardo. A realistic model for true concurrency. In *XII Jornadas de Concurrencia y Sistemas Distribuidos*, pages 13–26. Universidad Rey Juan Carlos, Junio 2004.

[RCCP05] M. Carmen Ruiz, Diego Cazorla, Fernando Cuartero, and Juan José Pardo. A formal specification and performance evaluation of the purchase phase in the set protocol. In *IEEE Symbolic and Numeric Algorithms for Scientific Computing*, pages 239–244, Los Alamitos, CA, USA, 2005. IEEE Computer Society.

[RCCP06a] M. Carmen Ruiz, Diego Cazorla, Fernando Cuartero, and Juan José Pardo. Analysis of the set e-commerce protocol using a true concurrency process algebra. In *21st ACM Symposium on Applied Computing (SAC-06)*, pages 879–886. ACM Press, 2006.

[RCCP06b] M. Carmen Ruiz, Diego Cazorla, Fernando Cuartero, and Juan José Pardo. A bounded true concurrency process algebra for automated verification. In *Electronic Notes in Theoretical Computer Science*, pages 1–10. Elsevier Science, 2006.

[RCCP06c] M. Carmen Ruiz, Diego Cazorla, Fernando Cuartero, and Juan José Pardo. Process algebra specification of flexible manufacturing systems. In *SYNASC*, pages 181–186, Los Alamitos, CA, USA, 2006. IEEE Computer Society.

[RCCP06d] M. Carmen Ruiz, Diego Cazorla, Fernando Cuartero, and Juan José Pardo. Specification and performance evaluation of flexible manufacturing systems using a bounded true concurrent process algebra. In *International Conference on Computational Intelligence for Modelling, Control and Automation (CIMCA/IAWTIC)*, page 46, Los Alamitos, CA, USA, 2006. IEEE Computer Society.

[Ros98] A.W. Roscoe. *The Theory and Practice of Concurrency*. Prentice-Hall Series in Computer Science. Prentice-Hall, 1998.

[RR88] G.M. Reed and A.W. Roscoe. A timed model for communicating sequential processes. *Theoretical Computer Science*, 58:249–261, 1988.

[RR99] G.M. Reed and A.W. Roscoe. The timed failures – stability model for CSP. *Theoretical Computer Science*, 211(1–2):85–127, 1999.

[Sch95] Steve Schneider. An operational semantics for timed csp. *Inf. Comput.*, 116(2):193–213, 1995.

[Sto01] S. D. Stoller. A Bound on Attacks on Payment Protocols. pages 61–70, 2001.

[SV89] M. Silva and R. Valette. Petri Nets and Flexible Manufacturing. *Advances in Petri Nets*, 424:374–417, 1989.

[TGVCE98] F. Tricas, F. Garcia-Valles, J.M. Colom, and J. Ezpeleta. A Structural approach to the problem of deadlock prevention in processes with resources. *proc. of WODES'98*, pages 273–278, 1998.

[Uza02] M. Uzam. An optimal deadlock prevention policy for Flexible Manufacturing Systems using Petri net models with resources and the theory of regions. *journal of Adv. Manuf. Tech.*, 19:192–208, 2002.

[Uza04] M. Uzam. The use of Petri net reduction approach for an optimal deadlock prevention policy for Flexible Manufacturing Systems. *Int. F. Adv. Manuf. Tech.*, 23:204–219, 2004.

[WNSS94] R.A. Wysk, Y. Neng-Shu, and J. Sanjay. Resolution of deadlocks in Flexible Manufacturing Systems: avoidance and recovery

approaches. *Journal of Manufacturing Systems*, 13(2):128–136–80, 1994.

[ZD93] M. Zhou and F. Dicesare. Petri Net Synthesis for Discrete Event Control of Manufacturing Systems. *Kluwer Academic Publishers*, 1993.

[ZDD89] M. Zhou, F. Dicesare, and A. Desreochers. A top-down Modular Approach to Synthesis of Petri Net models for Manufacturing Systems. *Proc. IEEE Robot. Automat. Conf*, pages 531–539, 1989.